大数据与人工智能技术丛书



Python机器学习

——数据分析与评分卡建模 微课版

◎ 翟锟 胡锋 周晓然 编著

本书特色

- **零基础入门，注重实战**

  10个学习实例，3个完整的项目案例

- **视频教学，全程语音讲解**

  270分钟高品质配套教学视频

- **教学资源丰富**

  提供教学课件、源代码、数据集

270分钟 微课视频

清华大学出版社

大数据与人工智能技术丛书

Python 机器学习
——数据分析与评分卡建模

（微课版）

翟锟　胡锋　周晓然　编著

清华大学出版社
北　京

内 容 简 介

本书在Python数据分析与建模方面,既是一本入门书,也是一本提高书,它提炼总结了作者从Python小白到Python建模工程师的历程。如果读者有志于数据分析、建模领域,那么它一定会带给读者惊喜。书中代码具有很高的可移植性,可供读者直接使用。

全书共分为8章,从Python的环境搭建到基本语法结构,从趣味应用到分析与建模,最后以社交网络分析结束。

本书附有教学视频、源代码、课件等配套资源,适用于银行业或互联网金融行业中的风控人员,金融行业中的数据分析师(或想转行数据分析师的学习者),以及正在学习机器学习的从业人员。

图书在版编目(CIP)数据

Python机器学习:数据分析与评分卡建模:微课版/翟锟,胡锋,周晓然编著.—北京:清华大学出版社,2019
(大数据与人工智能技术丛书)
ISBN 978-7-302-51684-2

Ⅰ.①P… Ⅱ.①翟… ②胡… ③周… Ⅲ.①软件工具—程序设计 Ⅳ.①TP311.561

中国版本图书馆CIP数据核字(2018)第264264号

责任编辑: 黄 芝
封面设计: 刘 键
责任校对: 时翠兰
责任印制: 沈 露

出版发行: 清华大学出版社
网　　址: http://www.tup.com.cn, http://www.wqbook.com
地　　址: 北京清华大学学研大厦A座　　**邮　　编:** 100084
社 总 机: 010-62770175　　**邮　　购:** 010-62786544
投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn
课件下载: http://www.tup.com.cn,010-62795954
印 刷 者: 北京富博印刷有限公司
装 订 者: 北京市密云县京文制本装订厂
经　　销: 全国新华书店
开　　本: 185mm×260mm　　**印　张:** 11.75　　**字　　数:** 200千字
版　　次: 2019年5月第1版　　**印　　次:** 2019年5月第1次印刷
印　　数: 1～2000
定　　价: 49.00元

产品编号:080057-01

前 言

Python自20世纪90年代初诞生以来，已逐渐被越来越多的开发者所接受甚至着迷。当然，这与其本身的简洁性、易读性以及可拓展性密不可分。Python具有丰富而强大的库，它能够很轻松地与其他语言的各种模块相结合。如果读者第一次见到用Python编写的语句，估计会被其整齐划一的设计风格所触动，Python的最大优点是易读、易维护。

本书以“零基础”为出发点，直接从实际应用案例入手，在内容方面更注重实战性。希望读者能在学习中不断思考，学以致用。

在本书的编写过程中，结合工作中的一些具体案例，整理成书。由于作者水平有限，疏漏在所难免，读者如发现问题，欢迎您及时指正。

特别声明，本书非系统性学习资料，内容的准备上有点跳跃，学习某些内容时还需要具备一些相关的基础知识，Google和百度会是阅读过程中的常伴。

为便于教学，本书有教学视频、源代码、课件等配套资源。

(1) 获取教学视频方式：读者可以先扫描本书封底的文泉云盘防盗码，再扫描书中相应的视频二维码，观看教学视频。

(2) 获取源代码、数据集方式：先扫描本书封底的文泉云盘防盗码，再扫描下方二维码，即可获取。



(3) 其他配套资源可以扫描封底课件二维码下载。

感谢本书的另外两名作者胡锋、周晓然，他们为此书同样牺牲了很多个人时间；感谢我的同事，他们在本书的写作过程中提供了很多的帮助；最后，感谢我的父母和妻子，把家庭生活安排得井井有条，让我能无后顾之忧，安心地编写此书！

希望这本书能够为正在学习或想要学习Python的读者提供帮助。

翟　锟

2019年1月

目 录

第 1 章

Python开发环境搭建

Python 是一种面向对象的解释型计算机程序设计语言，于 1989 年由荷兰人 Guido van Rossum 发明。1991 年，第一个公开的 Python 版正式发行。

近年来，随着机器学习、深度学习的快速发展，Python 的受欢迎程度越来越高，具体的排名顺序可以查看 TIOBE 官网：https://www.tiobe.com/tiobe-index/。2018 年 7 月 Python 的排名是第 4 名，如图 1-1 所示。

Apr 2018	Apr 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.777%	+0.21%
2	2		C	13.589%	+6.62%
3	3		C++	7.218%	+2.66%
4	5	^	Python	5.803%	+2.35%
5	4	˅	C#	5.265%	+1.69%
6	7	^	Visual Basic .NET	4.947%	+1.70%
7	6	˅	PHP	4.218%	+0.84%

图 1-1　TIOBE 2018 年 7 月编程语言排行榜

2018 年 3 月，Python 作者在邮件列表上宣布将于 2020 年 1 月 1 日终止支持 Python 2.7 版本。同时，由于越来越多的第三方库都不再维护 Python 2 版本，Python 3 是大势所趋。本书主要讲述 Python 3 的用法，有关 Python 2 的用法部分，

会做特殊说明。

工欲善其事，必先利其器。在实际开发 Python 代码的过程中，我们常用的“利器”有哪些呢？

本章只是简单地对 Windows 系统上相关 Python 开发软件的安装过程进行讲解，而 Mac OS 系统上的安装过程与之类似。Linux 系统上的安装，可以使用 Google 或者百度来查找相关的安装教程。在 Linux 系统上安装相关软件时，可能会遇到提示的错误信息情况，可在网上搜出相应的解决办法。

1.1　利器 1：Notepad 编辑器

Notepad(记事本)是代码编辑器或 Windows 中的小程序，用于文本编辑，是一款开源、小巧、免费的纯文本编辑器，具有运行便携、体积小、资源占用小，以及支持众多程序语言等优点。Notepad 支持的语言有 C++、C＃、Java 等主流程序语言，HTML、XML、Python、JavaScript 等网页/脚本语言。Notepad 内置支持多达 27 种语法高亮度显示，还可实现语法折叠、宏等常用功能。Notepad 的官网链接地址为：https://notepad-plus-plus.org/，对应的界面如图 1-2 所示。

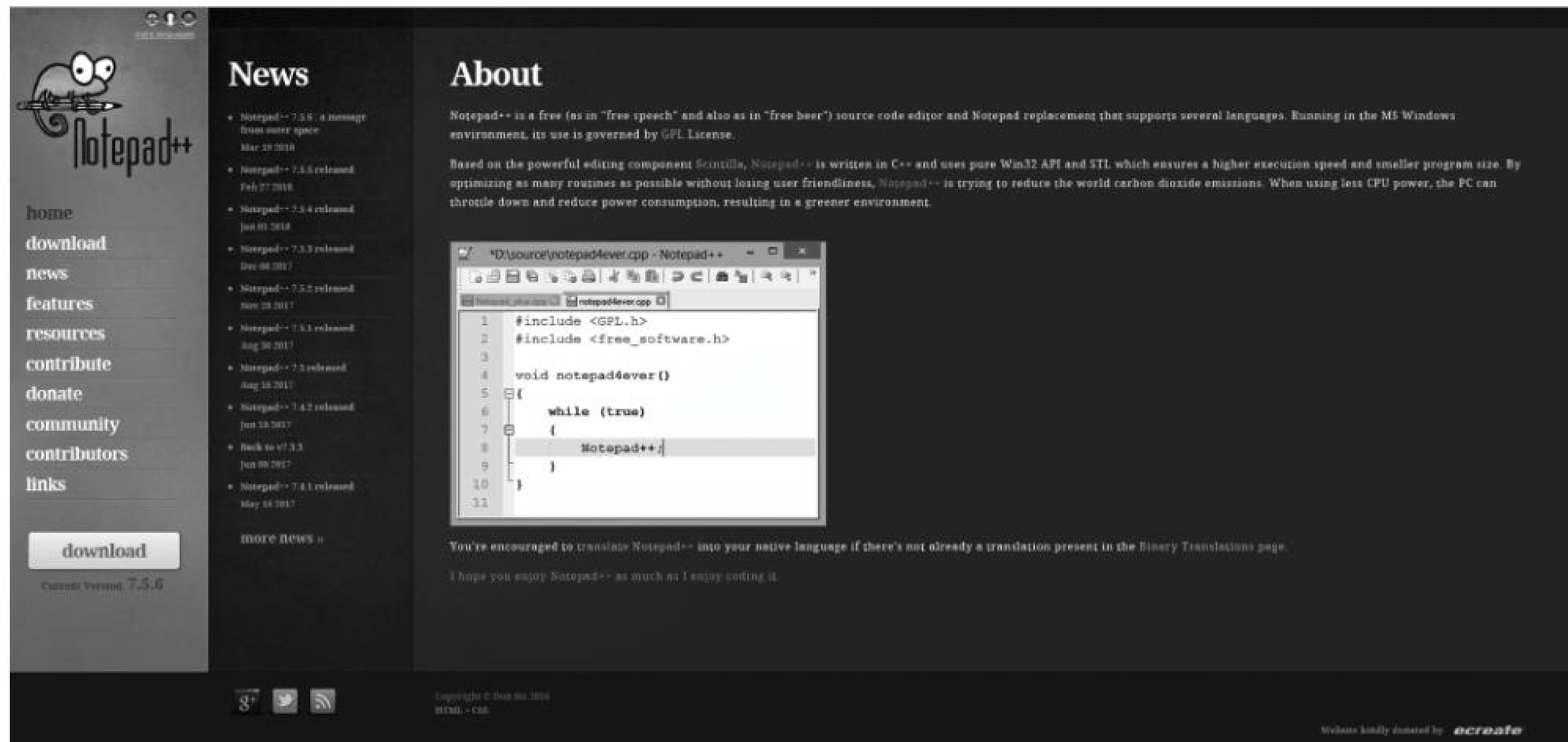


图 1-2　Notepad 官网首页界面图

安装好 Notepad 之后，其操作界面如图 1-3 所示。

Notepad 的优点主要包括以下 5 个方面。

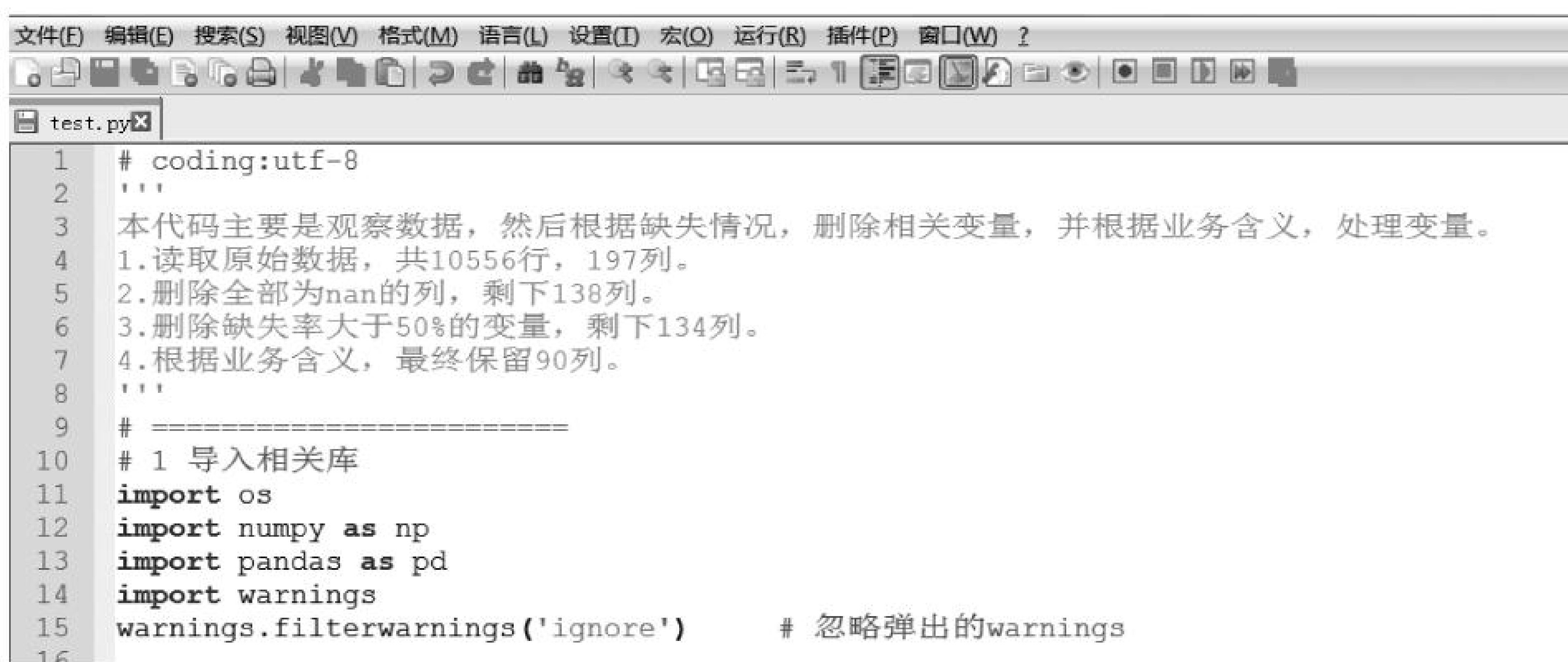


图 1-3 Notepad 界面图

(1) 语法高亮，具有字词自动完成功能，支持同时编辑多重文档；支持自定义语言。

(2) 自动检测文件类型，根据关键字显示节点，节点可自由折叠或打开，还可显示缩进引导线，使代码富有层次感。

(3) 在分窗口中又可打开多个子窗口，可以使用 F11 键快捷切换至全屏显示模式，支持鼠标滚轮改变文档显示比例。

(4) 可显示选中文本的字节数，并非普通编辑器所显示的字数；提供了一些实用工具，如邻行互换位置、宏功能等。

(5) 能够进行列块编辑，可快速地插入或者删除文本，能够成批替换文本。

1.2 利器 2：Anaconda

Anaconda 可以简单地理解为一个工具箱，其中包含了 conda、flask、nltk、pandas、pip 等 180 多个科学包及其依赖项，可以方便地实现包的安装、更新和卸载，比如机器学习包 scikit-learn、词云包 wordcloud 等。Anaconda 最大的好处在于它集成了 Jupyter Notebook 和 Spyder，这两个工具可以快速地让我们看到代码的运行结果，以便进行调试。

Anaconda 官网链接地址为：https://www.anaconda.com，官网首页如图 1-4 所示。Anaconda 下载链接地址为：https://www.anaconda.com/download/和 https://repo.continuum.io/archive/，如图 1-5 和图 1-6 所示。



图 1-4 Anaconda 官网首页界面图



图 1-5 Anaconda 下载界面图(1)

Anaconda installer archive

Filename	Size	Last Modified	MD5
Anaconda2-5.2.0-Linux-ppc64le.sh	269.6M	2018-05-30 13:04:31	479633a95906ea6d41056ebe84a4c47b
Anaconda2-5.2.0-Linux-x86.sh	488.7M	2018-05-30 13:05:30	758e172a824f467ea6b55d3d076c132f
Anaconda2-5.2.0-Linux-x86_64.sh	603.4M	2018-05-30 13:04:33	5c034a4ab36ec9b6ae01fa13d8a04462
Anaconda2-5.2.0-MacOSX-x86_64.pkg	616.8M	2018-05-30 13:05:32	2836c839d29be8d9569a715f4c631a3b
Anaconda2-5.2.0-MacOSX-x86_64.sh	527.1M	2018-05-30 13:05:34	b1f3fcf58955830b65613a4a8d75c3cf
Anaconda2-5.2.0-Windows-x86.exe	443.4M	2018-05-30 13:04:17	4a3729b14c2d3fccd3a050821679c702
Anaconda2-5.2.0-Windows-x86_64.exe	564.0M	2018-05-30 13:04:16	595e427e4b625b6eab92623a28dc4e21
Anaconda3-5.2.0-Linux-ppc64le.sh	288.3M	2018-05-30 13:05:40	cbd1d5435ead2b0b97dba5b3cf45d694
Anaconda3-5.2.0-Linux-x86.sh	507.3M	2018-05-30 13:05:46	81d5a1648e3aca4843f88ca3769c0830
Anaconda3-5.2.0-Linux-x86_64.sh	621.6M	2018-05-30 13:05:43	3e58f494ab9fbe12db4460dc152377b5
Anaconda3-5.2.0-MacOSX-x86_64.pkg	613.1M	2018-05-30 13:07:00	9c35bf27e9986701f7d80241616c665f
Anaconda3-5.2.0-MacOSX-x86_64.sh	523.3M	2018-05-30 13:07:03	b5b789c01e1992de55ee911754c310d4
Anaconda3-5.2.0-Windows-x86.exe	506.3M	2018-05-30 13:04:19	285387e7b6ea81edba98c011922e235a
Anaconda3-5.2.0-Windows-x86_64.exe	631.3M	2018-05-30 13:04:18	62244c0382b8142743622fdc3526eda7
Anaconda2-5.1.0-Linux-ppc64le.sh	267.3M	2018-02-15 09:08:49	e894dcc547a1c7d67deb04f6bba7223a
Anaconda2-5.1.0-Linux-x86.sh	431.3M	2018-02-15 09:08:51	e26fb9d3e53049f6e32212270af6b987
Anaconda2-5.1.0-Linux-x86_64.sh	533.0M	2018-02-15 09:08:50	5b1b5784cae93cf696e11e66983d8756
Anaconda2-5.1.0-MacOSX-x86_64.pkg	588.0M	2018-02-15 09:08:52	4f9c197dfe6d3dc7e50a8611b4d3cfa2
Anaconda2-5.1.0-MacOSX-x86_64.sh	505.9M	2018-02-15 09:08:53	e9845ccf67542523c5be09552311666e
Anaconda2-5.1.0-Windows-x86.exe	419.8M	2018-02-15 09:08:55	a09347a53e04a15ee965300c2b95dfde

图 1-6 Anaconda 下载界面图(2)

本节以 Anaconda3 5.0.0 为例，进行安装说明。

(1) 选中安装包，右击，再单击"以管理员身份运行"。

(2) 单击 Next 按钮，如图 1-7 所示。

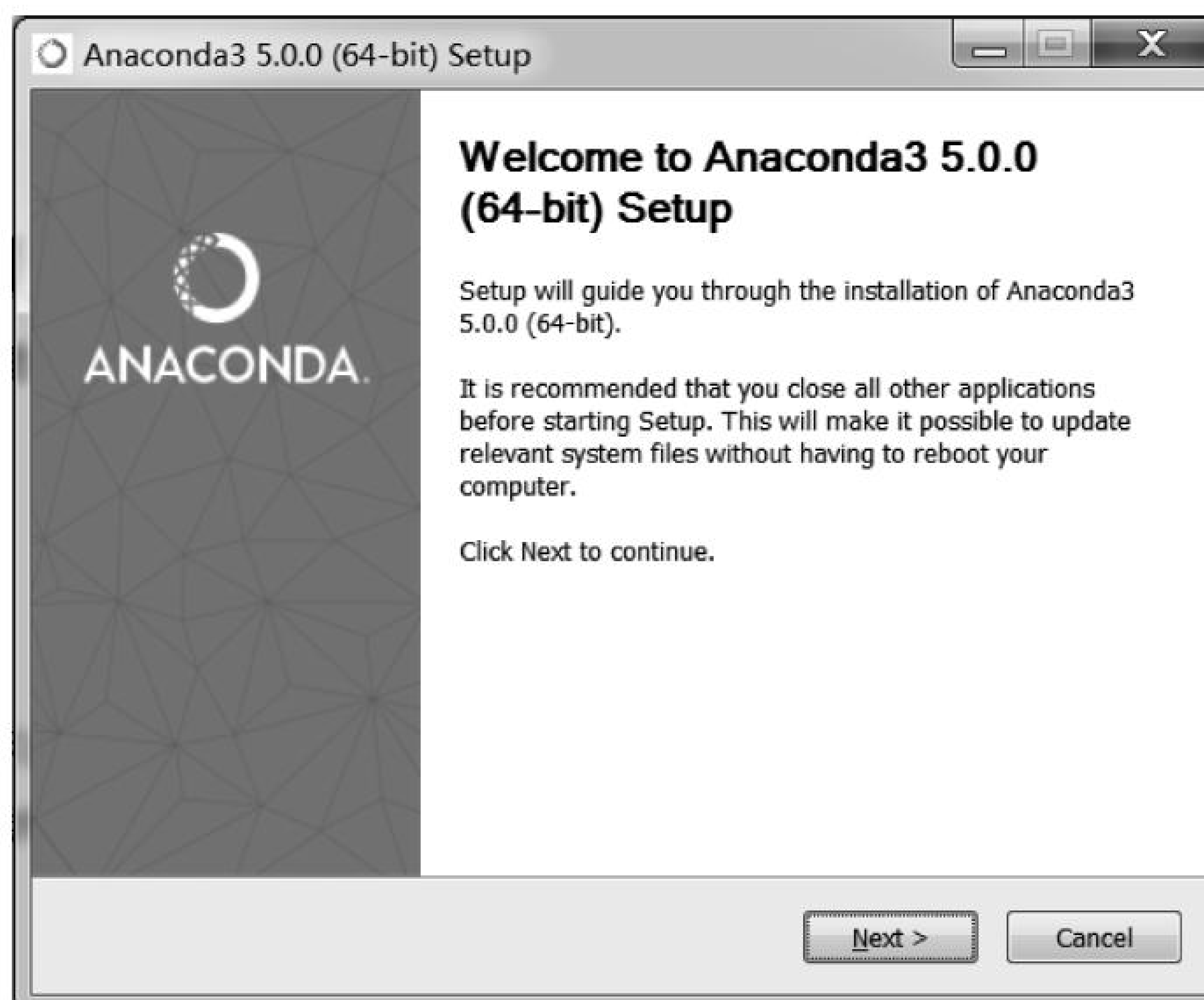


图 1-7　Anaconda3 5.0.0 安装界面图(1)

(3) 单击 I Agree 按钮，如图 1-8 所示。

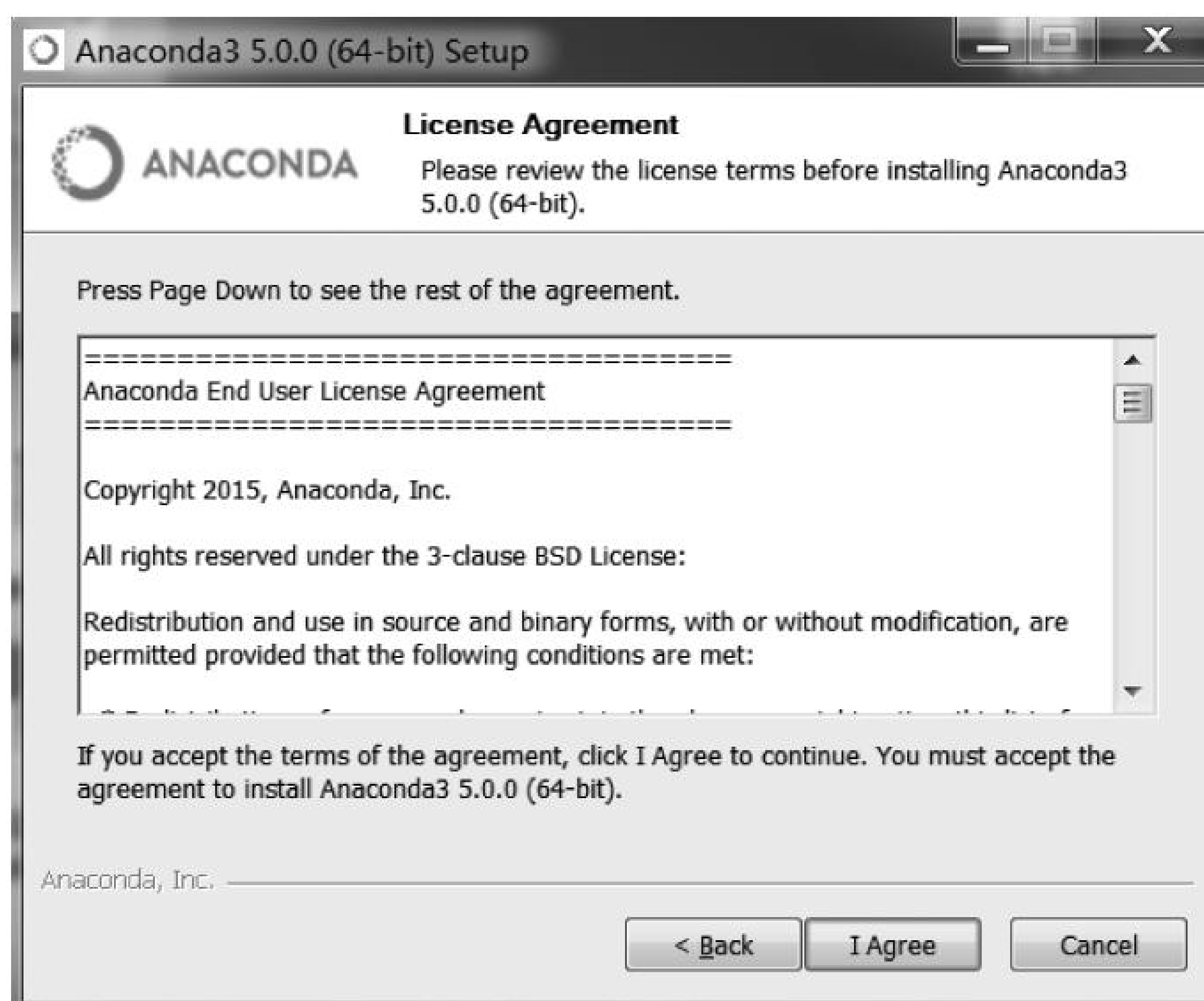


图 1-8　Anaconda3 5.0.0 安装界面图(2)

(4) 勾选 All Users(requires admin privileges),单击 Next 按钮,如图 1-9 所示。

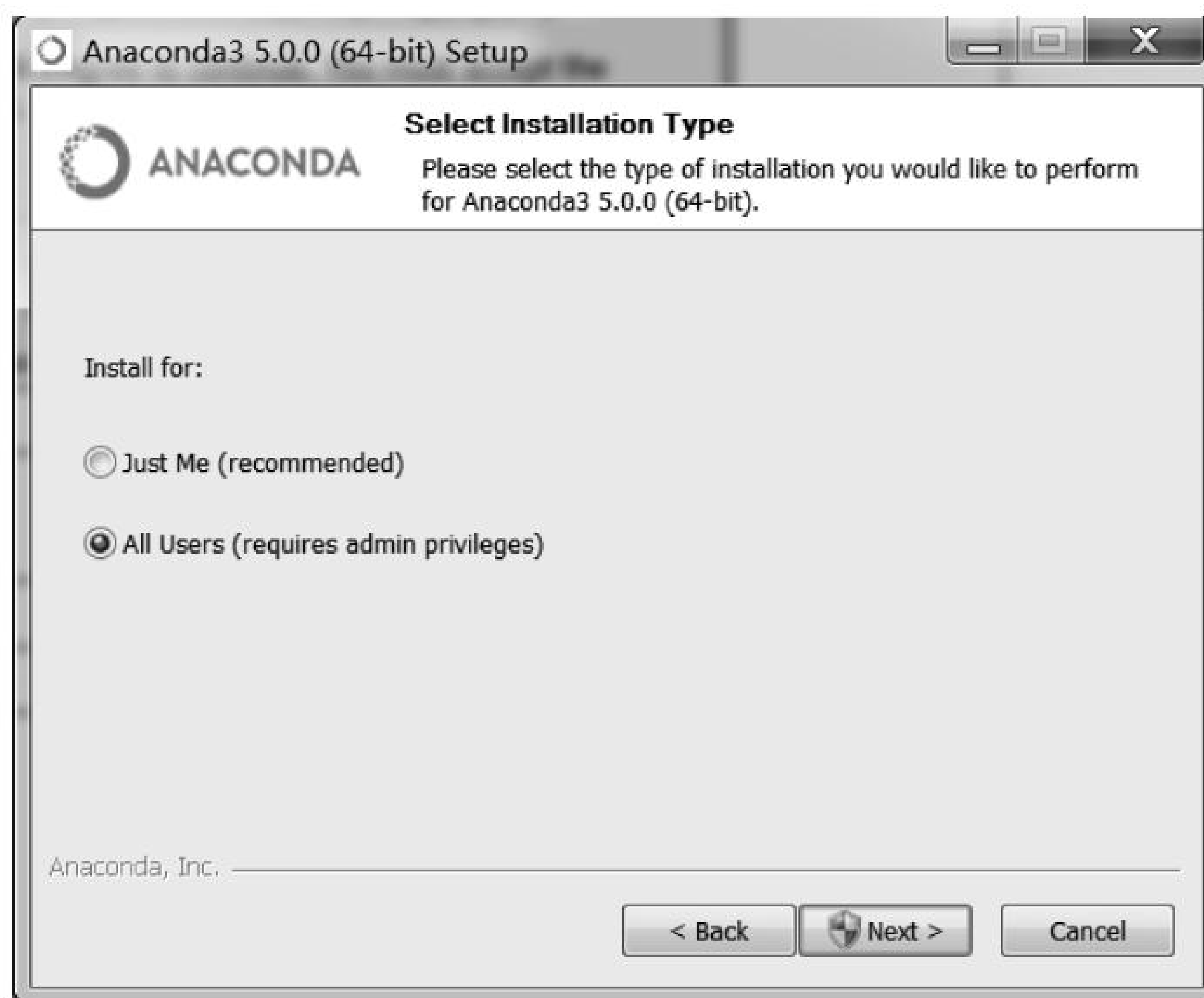


图 1-9 Anaconda3 5.0.0 安装界面图(3)

(5) 程序默认安装位置为 C:\ProgramData\Anaconda3,单击 Browse 可选择自定义安装目录,本安装教程的自定义安装路径为 D:\DevTools\Anaconda3,单击 Next 按钮,如图 1-10 所示。

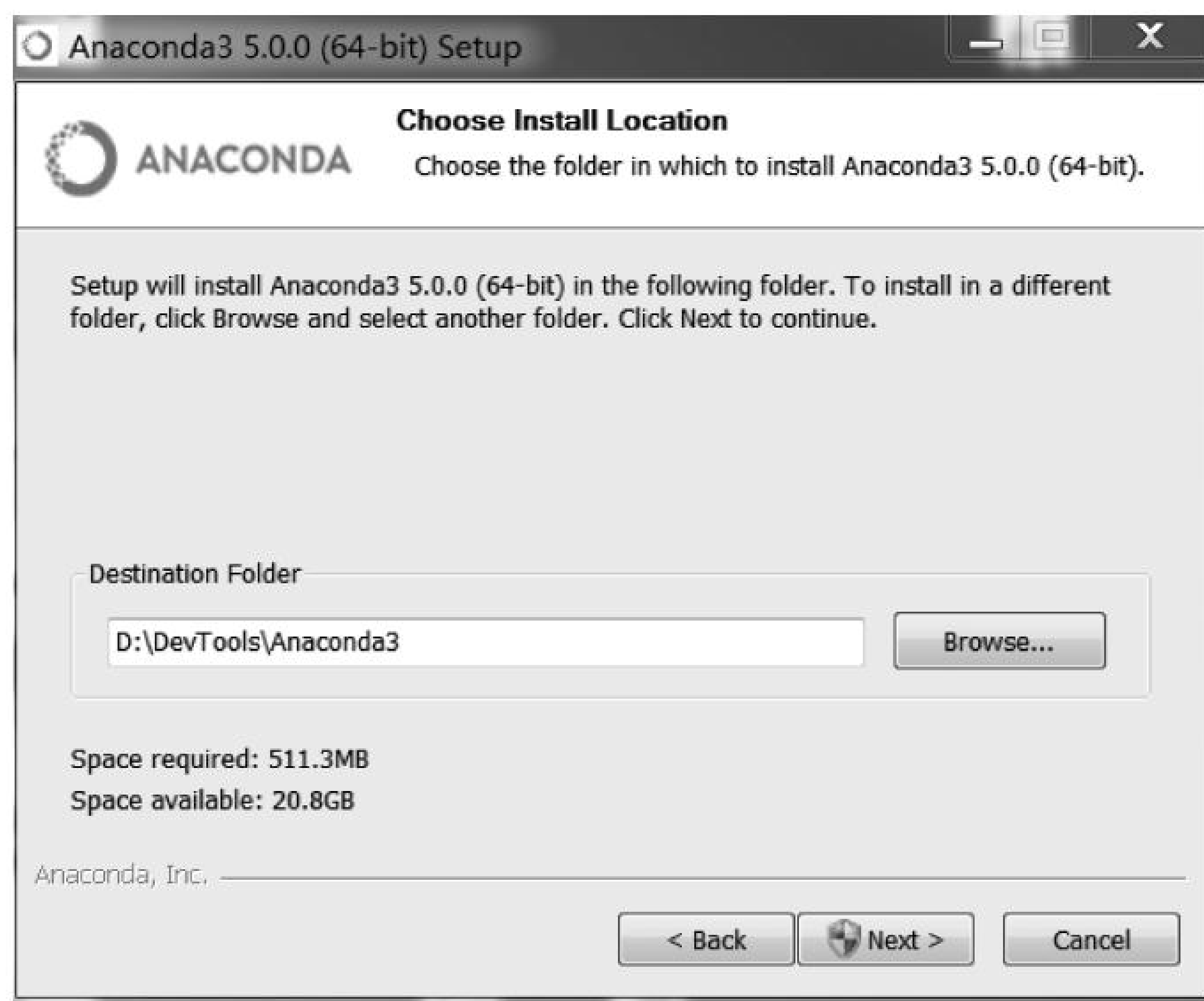


图 1-10 Anaconda3 5.0.0 安装界面图(4)

（6）同时勾选 Add Anaconda to the system PATH environment variable 和 Register Anaconda as the system Python 3.6，单击 Install 按钮，如图 1-11 所示。

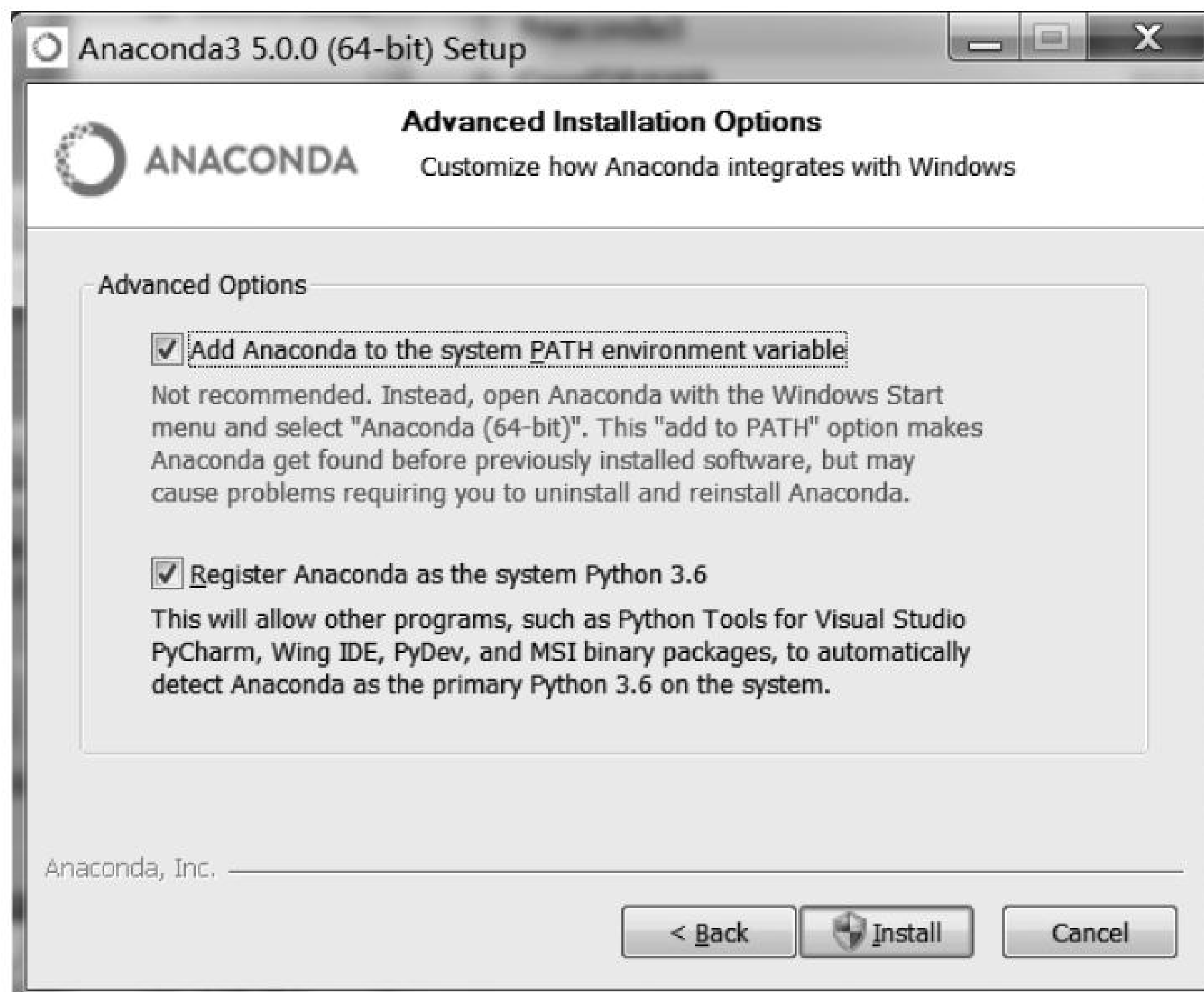


图 1-11　Anaconda3 5.0.0 安装界面图(5)

（7）安装过程需要 4～5 分钟，单击 Next 按钮，如图 1-12 所示。

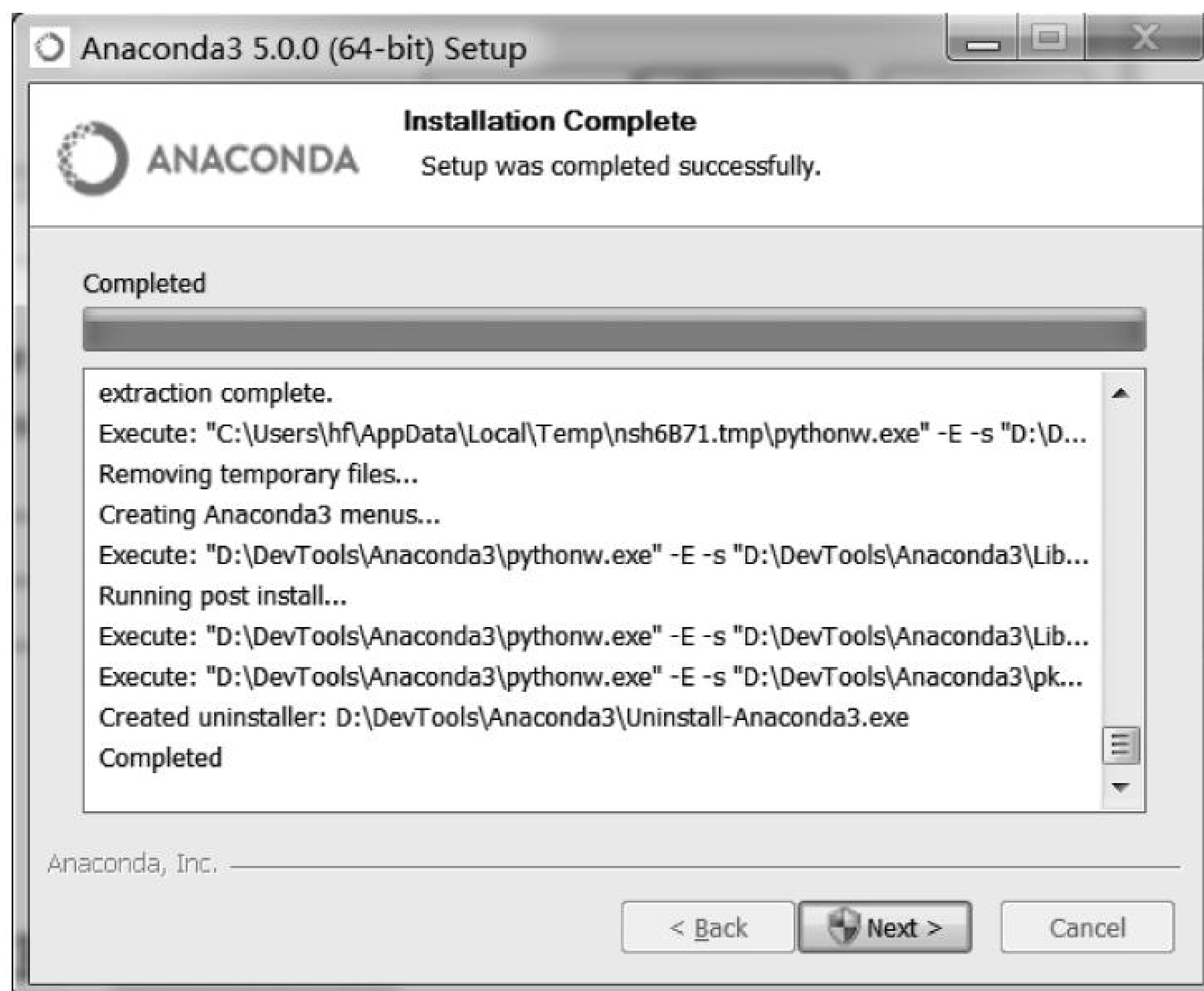


图 1-12　Anaconda3 5.0.0 安装界面图(6)

(8) 取消勾选 Learn more about Anaconda Cloud 和 Learn more about Anaconda Support，然后单击 Finish 按钮，至此完成了 Anaconda 的安装，如图 1-13 所示。

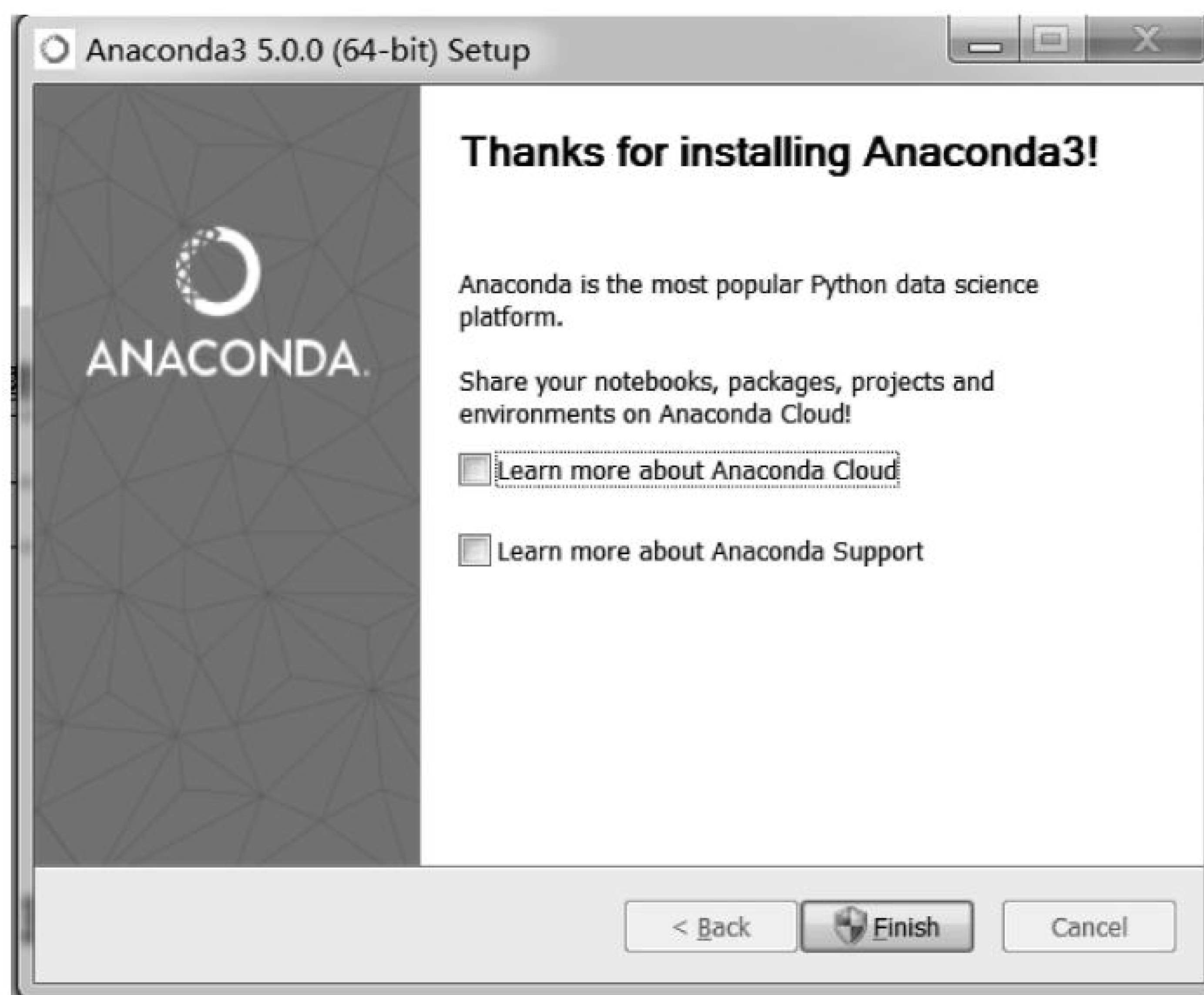


图 1-13　Anaconda3 5.0.0 安装界面图(7)

1.3　利器 3：Miniconda

Miniconda 相当于迷你版的 Anaconda，其功能比 Anaconda 稍微少一些。如果计算机存储空间比较小的话，可以安装 Miniconda，而不必安装 Anaconda。

Miniconda 官网链接地址为：https://conda.io/miniconda.html，如图 1-14 所示。Miniconda 下载链接地址为：https://repo.continuum.io/miniconda/，如图 1-15 所示。

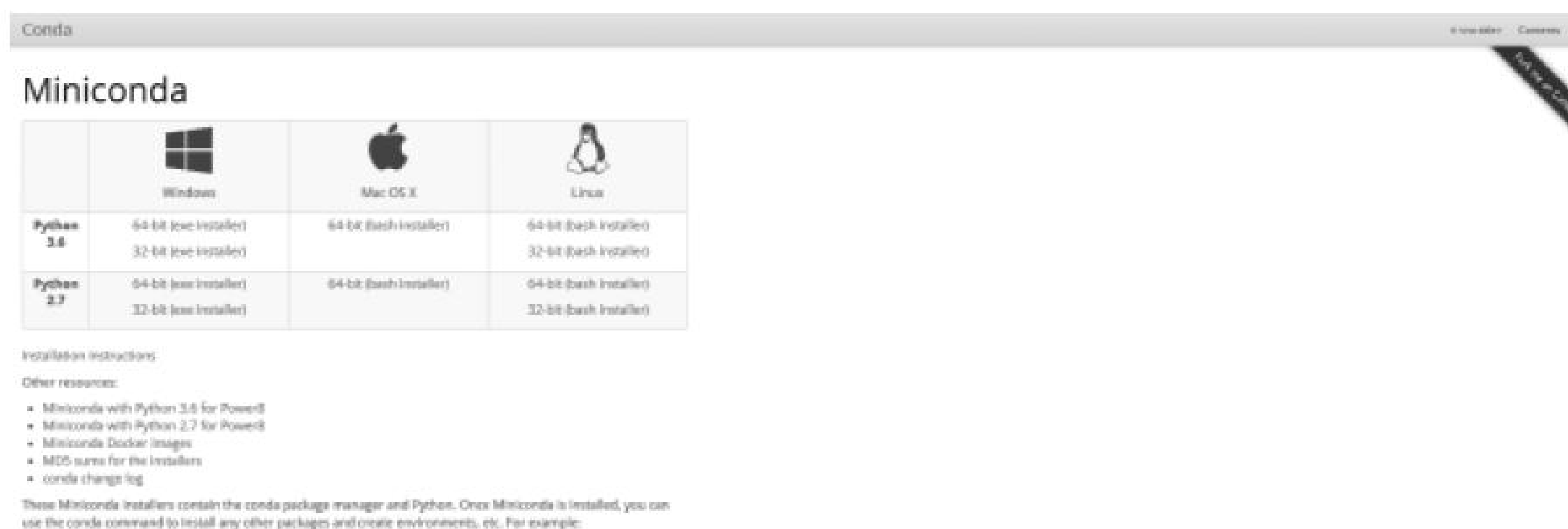


图 1-14　Miniconda 官网首页界面图

Miniconda installer archive

Filename	Size	Last Modified	MD5
Miniconda2-4.5.4-Linux-ppc64le.sh	36.9M	2018-06-06 23:07:18	3e26ee6447c8025609ea1e410f768417
Miniconda2-4.5.4-Linux-x86.sh	35.5M	2018-06-06 22:27:33	a638ae058a0ce15c5b289d151c488045
Miniconda2-4.5.4-Linux-x86_64.sh	38.1M	2018-06-06 22:24:38	8a1c02f6941d8778f8afad7328265cf5
Miniconda2-4.5.4-MacOSX-x86_64.pkg	34.5M	2018-06-06 23:12:27	6040ee82686b36f9bffa32d5f2a1341b
Miniconda2-4.5.4-MacOSX-x86_64.sh	29.8M	2018-06-06 23:12:26	35f4ca99d33ed56f68745eeaf1449274
Miniconda2-4.5.4-Windows-x86.exe	51.8M	2018-06-07 00:09:59	55502ce28ba3a16aa12954522d3624d2
Miniconda2-4.5.4-Windows-x86_64.exe	55.9M	2018-06-06 23:52:04	d285b7451deb4a33037033307c153684
Miniconda2-latest-Linux-ppc64le.sh	36.9M	2018-06-06 23:07:18	3e26ee6447c8025609ea1e410f768417
Miniconda2-latest-Linux-x86.sh	35.5M	2018-06-06 22:27:33	a638ae058a0ce15c5b289d151c488045
Miniconda2-latest-Linux-x86_64.sh	38.1M	2018-06-06 22:24:38	8a1c02f6941d8778f8afad7328265cf5
Miniconda2-latest-MacOSX-x86_64.pkg	34.5M	2018-06-06 23:12:27	6040ee82686b36f9bffa32d5f2a1341b
Miniconda2-latest-MacOSX-x86_64.sh	29.8M	2018-06-06 23:12:26	35f4ca99d33ed56f68745eeaf1449274
Miniconda2-latest-Windows-x86.exe	51.8M	2018-06-07 00:09:59	55502ce28ba3a16aa12954522d3624d2
Miniconda2-latest-Windows-x86_64.exe	55.9M	2018-06-06 23:52:04	d285b7451deb4a33037033307c153684
Miniconda3-4.5.4-Linux-ppc64le.sh	54.9M	2018-06-06 23:07:24	05c1e073f262105179cf57920dfc4d43
Miniconda3-4.5.4-Linux-x86.sh	53.7M	2018-06-06 22:27:35	0fcc79d640d82b7d36ea39654a82dd9d
Miniconda3-4.5.4-Linux-x86_64.sh	55.8M	2018-06-06 22:24:39	a946ea1d0c4a642ddf0c3a26a18bb16d
Miniconda3-4.5.4-MacOSX-x86_64.pkg	40.2M	2018-06-06 23:12:28	242882072fda2ada8551239e9041f5e9
Miniconda3-4.5.4-MacOSX-x86_64.sh	34.9M	2018-06-06 23:12:26	164ec263c4070db642ce31bb45d68813
Miniconda3-4.5.4-Windows-x86.exe	51.1M	2018-06-07 00:10:06	bc2f687a9e92455a099242929df8471d
Miniconda3-4.5.4-Windows-x86_64.exe	54.8M	2018-06-06 23:52:12	1c73051ccd997770288275ee6474b423
Miniconda3-latest-Linux-ppc64le.sh	54.9M	2018-06-06 23:07:24	05c1e073f262105179cf57920dfc4d43
Miniconda3-latest-Linux-x86.sh	53.7M	2018-06-06 22:27:35	0fcc79d640d82b7d36ea39654a82dd9d

图 1-15 Miniconda 下载界面图

Miniconda 的安装步骤与 Anaconda 的安装步骤类似，可参照 1.2 节中 Anaconda 的安装步骤，这里不再赘述。

1.4 利器 4：PyCharm IDE 工具

学过 Java、Go、Web 开发的，可能都知道 JetBrains 公司，该公司的主要产品有针对 Java 语言开发的 IntelliJ IDEA，针对 Go 语言开发的 GoLand，针对 Web 开发的 WebStorm，以及针对 Python 语言开发的 IDE 工具——PyCharm。

PyCharm 是一款 Python IDE，可以显著提高 Python 的开发效率，比如可以提高调试、语法高亮、Project 管理、代码跳转、智能提示、单元测试、版本控制等开发速率。在编写代码的过程中，PyCharm 可快速实现错误高亮，智能检测以及一键式代码快速补全建议，使得编码更加优化。

PyCharm 官网链接地址为：http://www.jetbrains.com/pycharm，如图 1-16 所示。下载链接地址为：http://www.jetbrains.com/pycharm/download/previous.html，如图 1-17 所示。

其具体的安装步骤，详见附录 A。

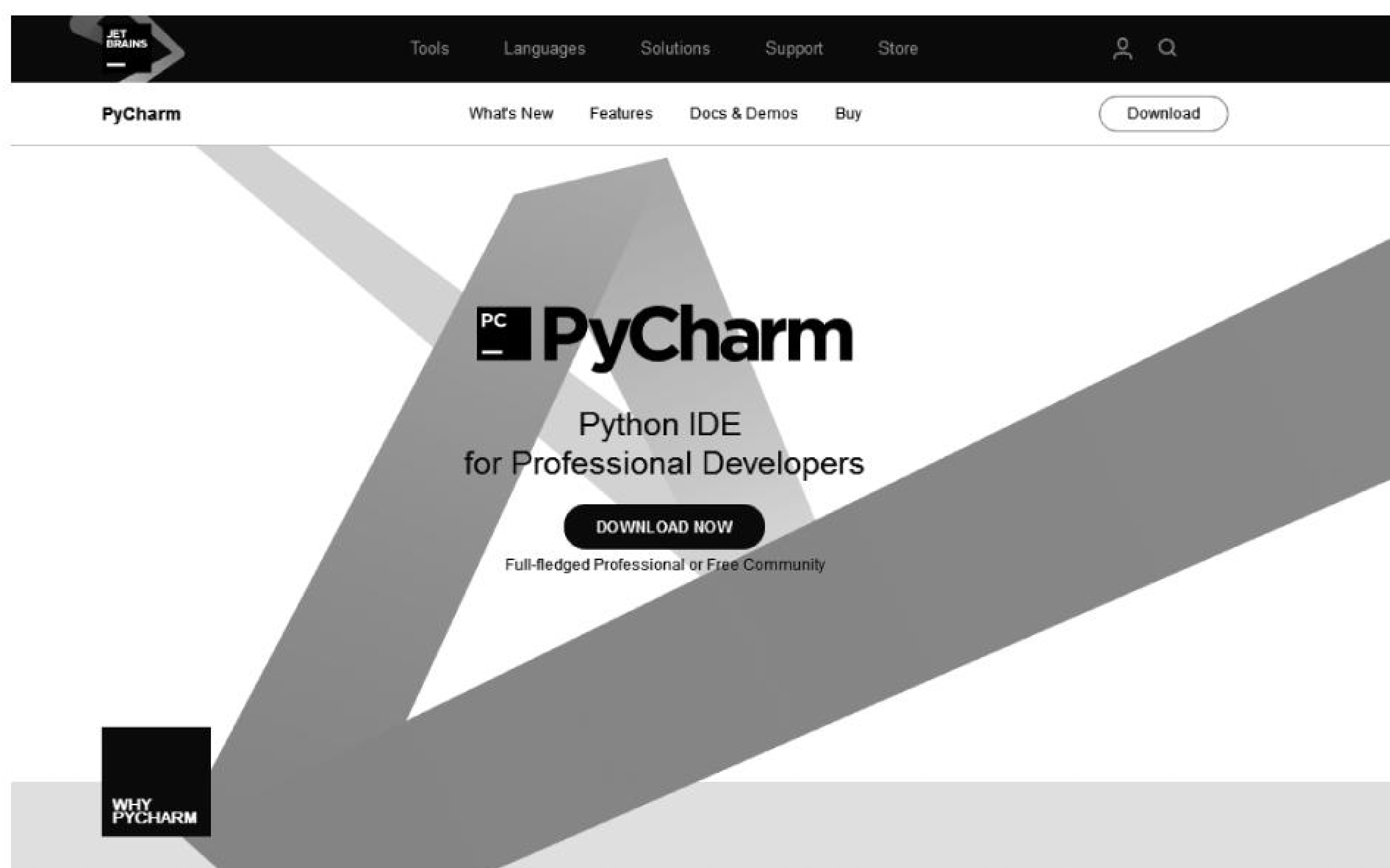


图 1-16 PyCharm 官网首页界面图

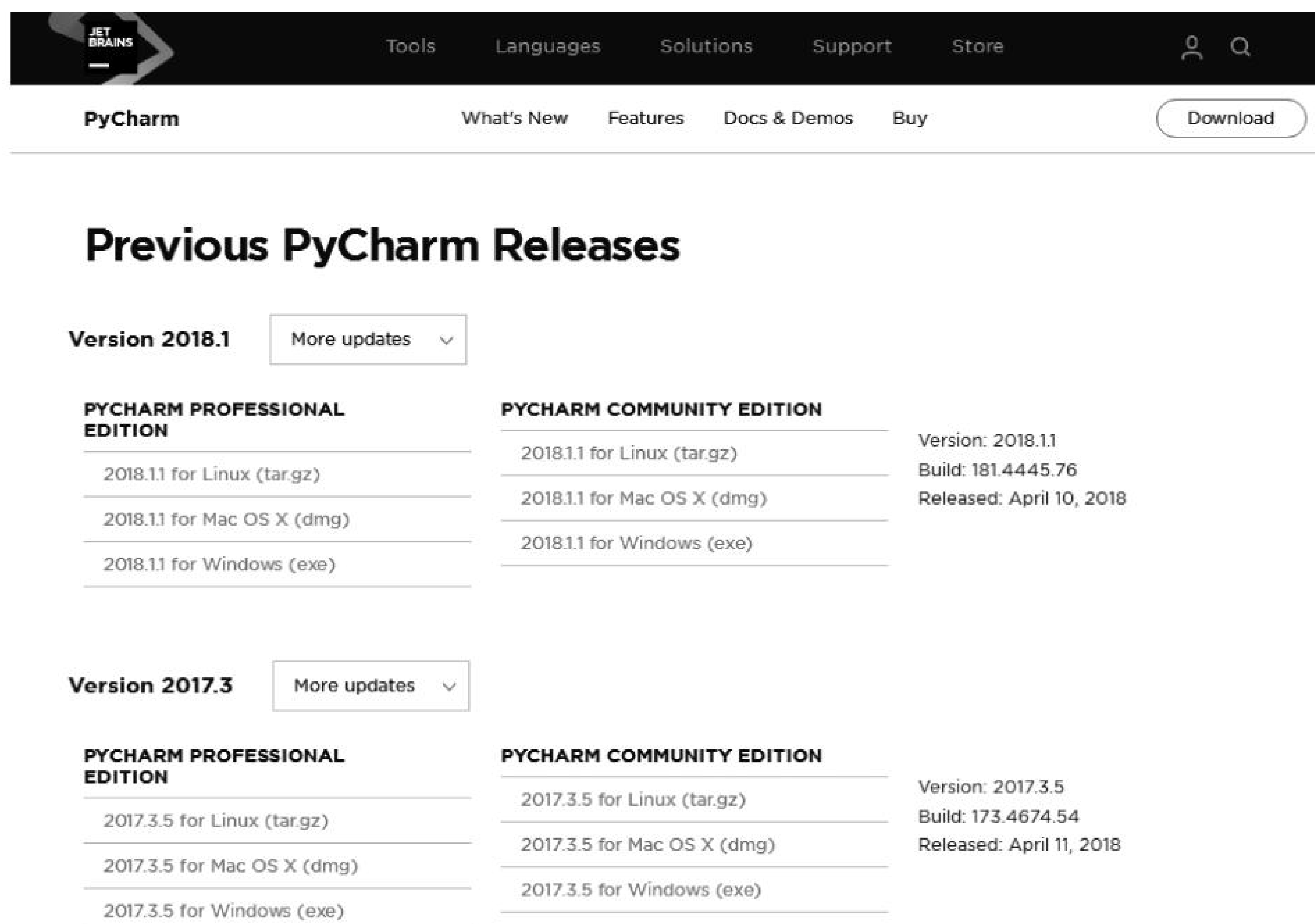


图 1-17 PyCharm 下载界面图

1.5　利器 5：Spyder

Spyder 是一个简单的集成开发环境，与 PyCharm 相比，具有更轻量级的特点，可以快速地查看代码运行的结果。Spyder 的界面是由许多窗格构成，用户可以根据自己的喜好调整它们的位置和大小。当多个窗格出现在一个区域时，将以标签页的形式显示。

Anaconda 安装包默认集成了 Spyder，安装好 Anaconda 后，就可以在 Anaconda 文件夹下查看到 Spyder。Spyder 的界面如图 1-18 所示，最左边是编写的代码，在右上角可以查看定义的变量，在右下角可以进行代码输出，看看代码的具体运行结果。

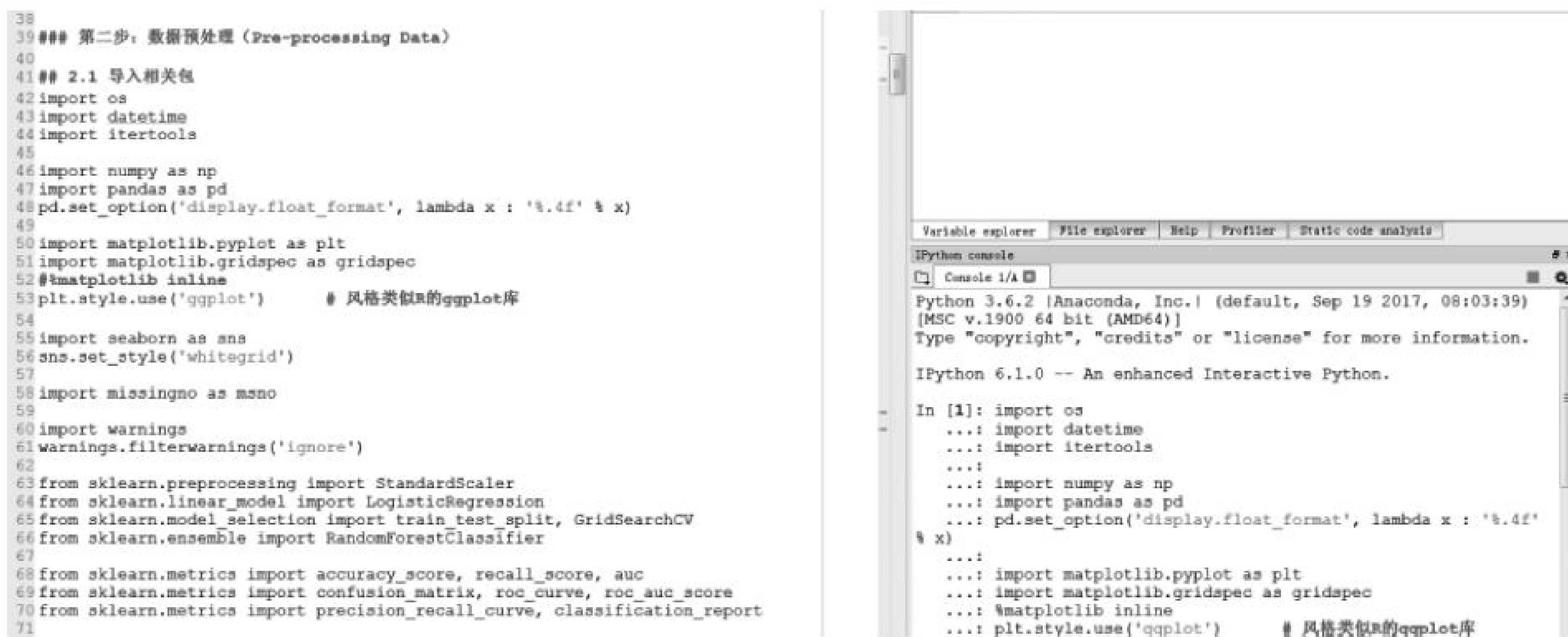


图 1-18　Spyder 界面图

1.6　利器 6：Jupyter Notebook

Jupyter Notebook（此前被称为 IPython notebook）是一个交互式笔记本，支持运行 40 多种编程语言。其本质是一个 Web 应用程序，便于创建和共享程序文档，支持实时代码，数学方程，可视化和 Markdown。

在编写 Python 代码的过程中，可以进行代码提示和结果显示。在机器学习中，Jupyter Notebook 常用于数据清洗、统计建模等。本书的代码主要是在 Jupyter Notebook 中进行编写和展现。

Anaconda 安装包默认集成了 Jupyter Notebook，安装好 Anaconda 后，就可以查看到 Jupyter Notebook。单击 New 按钮，选择 Python 3，就可以创建一个后缀为 .ipynb 的 Jupyter Notebook 文件，具体界面如图 1-19 所示。



图 1-19　创建 Jupyter Notebook 文件界面图

双击 Untitled，可以修改文件名称，本例中修改为 test，输入以下代码之后，按住 Shift＋Enter 键，就可以快速地查看输出结果并跳转到下一行，如图 1-20 所示。

```
import numpy as np
import pandas as pd
1 + 2
3 * 5
```

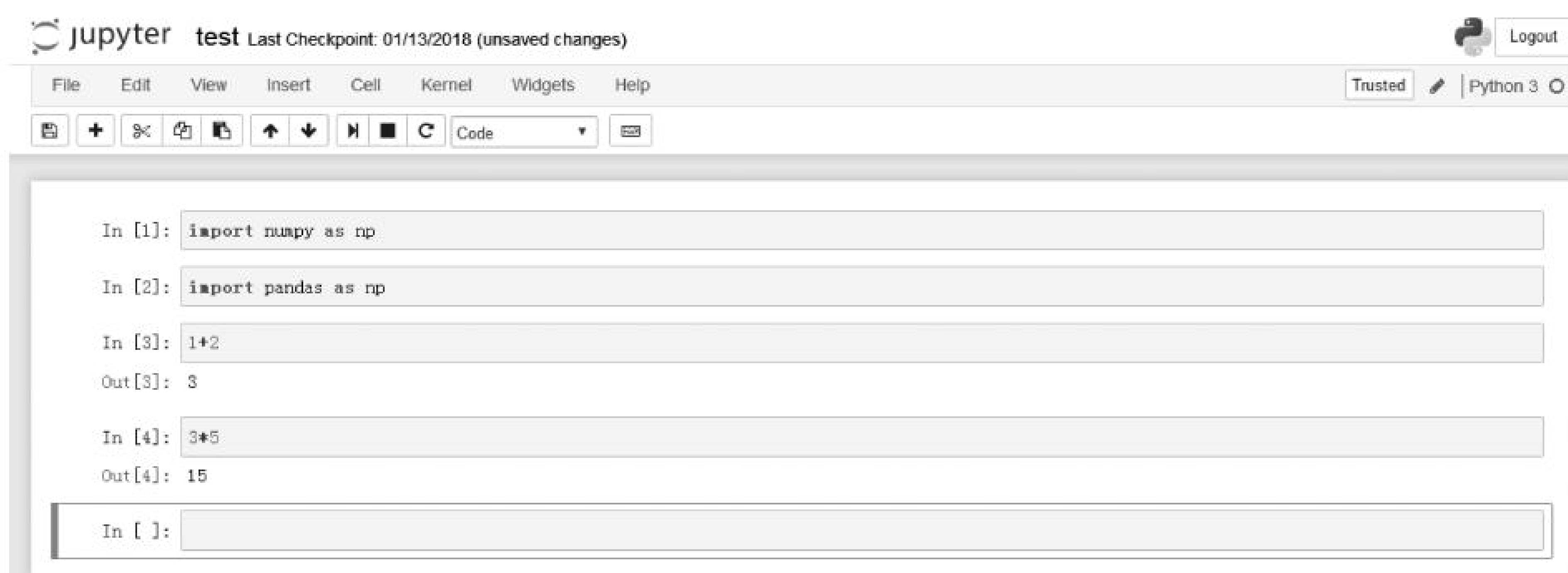


图 1-20　Jupyter Notebook 具体操作界面图

1.7　小结

本章主要介绍了在 Python 实际开发过程中，常用的编辑器、IDE 开发工具的安装和简介。关于 Linux 和 MacOS 系统上相关软件的安装过程，读者可以上网搜索相关的教程。

第 2 章

Python数据类型用法讲解

2.1 变量

在数学中，我们都学过二元一次方程组，比如：假设 x，y 满足方程 $x+y=3$，$x-y=1$，求 x 和 y 的值。这里求出的结果是 $x=2$，$y=1$。实际上，求出的 x 和 y 就是变量，只不过在 Python 中，很少用 x，y 来表示一个数值型变量，大部分都使用 i，j，k 来表示数值型变量。

先以 $x=2$ 为例，x 就是一个变量，这个变量存储了一个数值为 2 的值，这个 2 就是与变量相关的信息。通过等号"="，式中的 2 被赋值给了变量 x，即变量 x 保存了 2 在内存中的地址。同理，例如 msg= 'hello world!'，这里的 msg 就是一个变量。通过等号"="，'hello world!'被赋值给了 msg，即 msg 保存了'hello world!'在内存中的地址。

在使用变量时，良好的命名规范能让代码更容易阅读和理解，方便与项目组的其他成员进行交流。

以下是需要注意的有关变量的规则。

(1) 变量名只能包含字母、数字和下画线。变量名可以字母或下画线开头，但不

能以数字开头。例如,可将变量命名为 message_1,但不能将其命名为 1_message。

(2) 变量名不能包含空格,但可使用下画线来分隔其中的单词。例如,变量名 greeting_message 可行,但变量名 greeting message 会引发错误。

(3) 不要使用 Python 关键字和函数名用作变量名,Python 内置的 33 个关键字如表 2-1 所示,要想查看 Python 的关键字,可以输入 import keyword,然后再输入 keyword. kwlist 即可看到 Python 内置的 33 个关键字,如图 2-1 所示。

(4) 变量名应该简短,并且让程序员能够见名知意。例如,age 比 a 好,stu_name 比 s_n 好等。

(5) 慎用小写字母 l 和大写字母 O,因为它们可能被人错看成数字 1 和 0。

```
In [1]: import keyword
        keyword.kwlist

Out[1]: ['False',
         'None',
         'True',
         'and',
         'as',
         'assert',
         'break',
         'class',
         'continue',
         'def',
         'del',
         'elif',
         'else',
         'except',
         'finally',
         'for',
         'from',
         'global',
```

图 2-1 Python 内置的 33 个关键字列表

表 2-1 Python 内置的 33 个关键字

False	None	True	and	as	assert	break
class	continue	def	del	elif	else	except
finally	for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass	raise
return	try	while	with	yield		

要创建良好的变量名,需要经过一定的实践,在阅读源码的时候,可以学学源码的变量是怎么命名的,从而提高自己的编码能力。

2.2 字符串

在 Python 中,可以使用单引号‘’或者双引号“”来创建字符串。在创建字符串时,只要为变量分配一个值即可。例如:

```
str1 = 'hello world!'
str2 = "hello python!"
```

同时,单引号‘’和双引号“”也可以结合使用,即可以在字符串中包含引号和撇号。例如:

```
'I told my friend, "Python is my favorite language!"'
"One of Python's strengths is its diverse and supportive community."
```

那么,了解字符串的定义之后,该如何对字符串进行操作呢?此时就需要先知道字符串的下标和切片。

所谓的下标,可以理解为编号,就好比超市存储柜的编号、学生名单表中的学号,通过这个编号或者学号就能找到相应的存储空间或者学生信息。需要注意的是,在 Python 中,下标是从 0 开始的,不是从 1 开始的。

字符串实际上就是字符的数组,所以支持下标索引。如:msg="吃饭了吗?",msg[0]="吃",msg[1]="饭",以此类推。相似地,name='hello',则 name[0]='h',name[1]='e',以此类推。

切片是指对操作的对象截取其中一部分的操作。字符串、列表、元组都支持切片操作。切片语法:[起始:结束:步长]。选取的区间属于左闭右开型,即从“起始”位开始,到“结束”位的前一位结束(不包含结束位本身)。对应到数学中,就是前闭后开型区间,也可以理解为 start <= i < end。

例如:name='hello',则切片的常见操作如表 2-2 所示。

表 2-2 切片的常见操作示例

字符串切片语法	含 义
name[0:3] ='hel'	取下标 0~2 的字符
name[0:5] ='hello'	取下标 0~4 的字符
name[3:5] ='lo'	取下标 3、4 的字符
name[2:] ='llo'	取下标为 2 开始到最后的字符
name[1:-1] ='ell'	取下标为 1 开始到倒数第 2 个之间的字符
name[::2] ='hlo'	从下标 0 开始到最后字符,每隔 2 个位置,取出字符
name[5:1:-2] ='ol'	从下标 5 开始,到下标 2 逆序,每隔 2 个位置,取出字符

在实际操作过程中,如果不知道某些函数的具体用法,则可以使用 help 命令来进行提示和理解。比如,想知道字符串 str 的简介,那么可以在 Jupyter Notebook 中,输入 help(str)命令,就可以看到与字符串有关的介绍;输入 dir(str)命令,则会显示出字符串所有的私有方法和公有方法(在 Python 中,以两个下画线开头的方法就默

认为私有方法)。对于字符串中的某一特定方法,比如 find 的用法,可以输入 help(str.find)命令,就可以看到 find 方法的具体介绍和用法,包括函数的传入参数以及函数返回值,如图 2-2、图 2-3 和图 2-4 所示。

```
In [2]: help(str)

        Help on class str in module builtins:

        class str(object)
         |  str(object='') -> str
         |  str(bytes_or_buffer[, encoding[, errors]]) -> str
         |
         |  Create a new string object from the given object. If encoding or
         |  errors is specified, then the object must expose a data buffer
         |  that will be decoded using the given encoding and error handler.
         |  Otherwise, returns the result of object.__str__() (if defined)
         |  or repr(object).
         |  encoding defaults to sys.getdefaultencoding().
         |  errors defaults to 'strict'.
         |
         |  Methods defined here:
         |
         |  __add__(self, value, /)
         |      Return self+value.
```

图 2-2　help(str)的用法

```
In [3]: dir(str)

Out[3]: ['__add__',
         '__class__',
         '__contains__',
         '__delattr__',
         '__dir__',
         '__doc__',
         '__eq__',
         '__format__',
         '__ge__',
         '__getattribute__',
         '__getitem__',
         '__getnewargs__',
         '__gt__',
         '__hash__',
         '__init__',
         '__init_subclass__',
         '__iter__',
         '__le__',
         '__len__',
         '__lt__',
         '__mod__',
```

图 2-3　dir(str)的用法

下面介绍字符串的常用操作。

字符处理类函数及用法如表 2-3 所示。

```
In [4]: help(str.find)

Help on method_descriptor:

find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

图 2-4 help(str.find)的用法

表 2-3 字符处理类函数及用法

函数名及含义	例 子

str.upper()
将所有小写字母变为大写(只对字母起作用)
	>>> str1 = 'HELLO world 123'
	>>> str1.upper()
	'HELLO WORLD 123'

str.lower()
将所有大写字母变为小写(只对字母起作用)
	>>> str1 = 'HELLO world 123'
	>>> str1.lower()
	'hello world 123'

str.swapcase()
所有大小写字母互换
	>>> str1 = 'HELLO world 123'
	>>> str1.swapcase()
	'hello WORLD 123'

str.capitalize()
首字母大写,其余字母小写(要保证字符串的第一个元素是字母)
	>>> str1 = 'HELLO world 123'
	>>> str1.capitalize()
	'Hello world 123'
	>>> str2 = '3HELLO world'
	>>> str2.capitalize()
	'3hello world'

str.title()
字符串的每个单词首字母大写
	>>> str1 = 'HELLO world 123'
	>>> str1.title()
	'Hello World 123'
	>>> str2 = '123hello world'
	>>> str2.capitalize()
	'123Hello World'

字符串搜索相关类函数及用法如表 2-4 所示。

表 2-4　字符串搜索相关类函数及用法

函数名及含义	例　　子

str.find(substr,start,end)
检测字符串 str 是否包含子字符串 substr。如果包含,返回子字符串 substr 开始的位置索引值,否则返回－1
	>>> str1 = 'HELLO world 123'
	>>> str1.find('wo')
	6
	>>> str1.find('wo', 10)
	－1

str.rfind(substr,start,end)
类似于 find()函数,不过是从右边开始查找 substr
	>>> str1 = 'HELLO world 123'
	>>> str1.rfind('L')
	3

str.index(substr,start,end)
与 find()方法一样,如果 substr 不在字符串中则会报异常
	>>> str1 = 'HELLO world 123'
	>>> str1.index('wo')
	6
	>>> str1.index('wo',10)
	ValueError: substring not found

str.rindex(substr,start,end)
类似于 index(),不过是从右边开始查找 substr
	>>> str1 = 'HELLO world 123'
	>>> str1.rindex('L')
	3
	>>> str1.rindex('h')
	ValueError: substring not found

str.count(substr,start,end)
计算子字符串 substr 在字符串 str 中出现的次数
	>>> str1 = 'HELLO world 123'
	>>> str1.count('L')
	2
	>>> str1.count('z')
	0

字符串判断类函数及用法如表 2-5 所示。

表 2-5　字符串判断类函数及用法

函数名及含义	例　　子

str.startswith(substr)
检查字符串是否是以 substr 开头,是则返回 True,否则返回 False
	>>> str1 = 'HELLO world 123'
	>>> str1.startswith('HE')
	True
	>>> str1.startswith('he')
	False

str.endswith(substr)
检查字符串是否以 substr 结束,如果是返回 True,否则返回 False
	>>> str1 = 'HELLO world 123'
	>>> str1.endswith('123')
	True
	>>> str1.endswith('456')
	False

续表

函数名及含义	例　　子

str.isupper()
检查字符串里的字母是否都是大写字母(注意：只检查字母,不检查空格、数字等)
	>>> str1 = 'HELLO world 123'
	>>> str1.isupper()
	False
	>>> str2 = 'HELLO WORLD 123'
	>>> str2.isupper()
	True

str.islower()
检查字符串里的字母是否都是小写字母(注意：只检查字母,不检查空格、数字等)
	>>> str1 = 'hello world 123'
	>>> str1.islower()
	True

str.isalpha()
检查字符串是否全由字母组成,如果是返回True,否则返回 False。(注意字符串中是否含有空格)
	>>> str1 = 'HELLOworld'
	>>> str1.isalpha()
	True
	>>> str2 = 'HELLO world'
	>>> str2.isalpha()
	False

str.isdigit()
检查字符串是否全由数字组成,如果是返回True,否则返回 False。
	>>> str1 = '123'
	>>> str1.isdigit()
	True
	>>> str2 = '12 3'
	>>> str2.isdigit()
	False
	>>> str3 = '123!'
	>>> str3.isdigit()
	False

str.isalnum()
检查字符串是否全为字母或数字
	>>> str1 = 'hello123'
	>>> str1.isalnum()
	True
	>>> str2 = 'hello 123'
	>>> str2.isalnum()
	False
	>>> str3 = 'hello123!'
	>>> str3.isalnum ()
	False

续表

函数名及含义	例 子

str.isspace()
检查字符串中是否只包含空格,如果是则返回 True,否则返回 False
	>>> str1 = ''
	>>> str1.isspace()
	False
	>>> str2 = ' '
	>>> str2.isspace()
	True
	>>> str3 = ' '
	>>> str3.isspace()
	True

字符串格式化类函数及用法如表 2-6 所示。

表 2-6 字符串格式化类函数及用法

函数名及含义	例 子

str.ljust(width)
返回一个原字符串左对齐,并使用空格填充至长度为 width 的新字符串
	>>> str1 = 'hello'
	>>> str1.ljust(10)
	'hello '(此处有 5 个空格)

str.rjust(width)
返回一个原字符串右对齐,并使用空格填充至长度为 width 的新字符串
	>>> str1 = 'hello123'
	>>> str1.rjust(10)
	' hello123'(此处有两个空格)

str.center(width)
返回一个原字符串居中,并使用空格填充至长度为 width 的新字符串(当字符串不可能居中对齐时,左边的空格会比右边的少一个)
	>>> str1 = 'hello'
	>>> str1.center(7)
	' hello '(左右两边各一个空格)
	>>> str1.center(8)
	' hello '(左边一个空格,右边两个空格)

字符串其他类函数及用法如表 2-7 所示。

表 2-7 字符串其他类函数及用法

函数名及含义	例 子

replace(str1, str2, count)
用 str2 替换掉 str1,如果 count 指定,则替换不超过 count 次
	>>> name = 'hello world world'
	>>> name.replace('world', 'Python')
	'hello Python Python '
	>>> name.replace('world', 'Python', 1)
	'hello Python world'

续表

函数名及含义	例　　子

split(str=' ', maxsplit)
以 str 为分隔符切片字符串,如果 maxsplit 有指定值,则仅分隔 maxsplit 个子字符串
	>>> name = 'hello world Python Go'
	>>> name.split(' ')
	['hello', 'world', 'Python', 'Go']
	>>> name.split(' ', 2)
	['hello', 'world', 'Python Go']

strip()
删除字符串两端的空白字符
	>>> mystr = '\n\t hello \t\n'
	>>> mystr.strip()
	'hello'

lstrip()
删除字符串左边的空白字符
	>>> mystr = ' hello'
	>>> mystr.lstrip()
	'hello'
	>>> mystr = ' hello '
	>>> mystr.lstrip()
	'hello '

rstrip()
删除字符串末尾的空白字符
	>>> mystr = ' hello '
	>>> mystr.rstrip()
	' hello'

partition(str)
把字符串以 str 分割成三部分,str 前、str 和 str 后,并保留 str,返回的是一个元组 tuple
	>>> mystr = 'hello world Python and Go'
	>>> mystr.partition('Python')
	('hello world ', 'Python', ' and Go')

rpartition(str)
类似于 partition()函数,不过是从右边开始
	>>> mystr = 'hello world Python and Go'
	>>> mystr.rpartition('and')
	('hello world Python ', 'and', 'Go')

splitlines()
按照行分割,返回一个包含各行作为元素的列表
	>>> mystr = 'hello\nworld'
	>>> print(mystr)
	hello
	world
	>>> mystr.splitlines()
	['hello', 'world']

join(str)
字符串中每个字符后面插入 str,构造出一个新的字符串
	>>> str = ' '
	>>> li = ['my', 'name', 'is', 'Python']
	>>> str.join(li)
	'my name is Python'
	>>> str = '_'
	>>> str.join(li)
	'my_name_is_Python'

看完上述字符串常用操作后，可以试着理解一下这些函数的可能实现方法。比如：判断一个字符串是否是以字母“a”开头，或者是否以字母“c”结束。首先可能会想到使用 startswith(或者 endswith)函数，但对于一名初学者来说，如果只接触过基本的下标或者切片，能不能实现该函数的功能呢？

以“a”开头，取下标为 0 的字符，与字母“a”进行比较，如果相等，则返回 True，否则返回 False。同理，取出字符串的最后一个字符，与字母“c”进行比较，如果相等则返回 True，如果不相等则返回 False。

但是在实际开发过程中，可能要考虑更多的情况。假设传进来的参数不是一个字符串，该怎样处理？如果是字符串，会不会是空字符串呢？字符串的长度是否大于要比较的字符串的长度等情况，都需要考虑到。

下面简单写一个判断字符串是不是以某一特定字符串开头的函数(默认传进来的参数都是字符串类型)。如果是，则返回 True；否则返回 False。其代码运行结果如图 2-5所示。

```
def startswith_prefix(mystr, prefix):
    if len(mystr) >= len(prefix) and int(mystr[:len(prefix)] == prefix):
        return True
    else:
        return False
```

```
In [5]: def startswith_prefix(mystr, prefix):
            if len(mystr) >= len(prefix) and int(mystr[:len(prefix)] == prefix):
                return True
            else:
                return False

In [6]: startswith_prefix('python','py')
Out[6]: True

In [7]: startswith_prefix('python','go')
Out[7]: False
```

图 2-5 startswith_prefix 函数运行结果

希望读者在学习字符串的常用操作时，多思考哪些基本的操作能够组合起来，从而实现一个内置函数的用法。也希望读者在面对一个项目问题时，能够快速地想到一些内置函数，以提高自己的代码质量，从而减轻自己的测试负担。

2.3 列表 list

列表 list 是一种有序的集合,可以随时添加和删除其中的元素,其长度是可变的。在 Python 中,使用中括号[ ]来表示一个列表,列表中的元素可以是 int 型、string 型,也可以是 int 和 string 的混合型。比如,列出班级里所有学生的名字,就可以用一个 list 表示:

```
classmates = ["张三", "李四", "王五"]
```

变量 classmates 就是一个列表,可以使用下标访问列表元素的值,也可以修改列表的值。其中,classmates[0]的输出结果就是"张三"。

使用 len()函数可以获得 list 元素的个数:len(classmates)的输出结果为 3。注意,列表中的元素是可以重复的,并且列表 list 的索引是从 0 开始的。当访问的索引值超出列表长度时,就会报错,提示 IndexError: list index out of range。所以要确保索引不要越界,并且列表的最后一个元素的索引是 len(classmates)-1。

下面介绍列表的常见操作。

2.3.1 增(append、insert、extend)

列表 list 是一个长度可变的有序表,可往 list 中动态地添加元素。有两种追加方法,一种是在列表的末尾添加元素,另一种是在列表的指定位置添加元素。

例如:假设编程语言列表 language=["python", "go", "java"],现在想把 C 语言和 PHP 添加到编程语言列表的末尾,使用 append 或者 extend。

当通过 append 方法向列表中添加新元素时,新元素的数据格式不会发生改变。但当使用 extend 方法向列表中添加新元素时,如果新元素也是一个列表,则会循环遍历该列表中的元素,将其添加到原始列表中。具体用法和不同之处,如图 2-6 所示。

insert 方法是在指定的索引位置处,直接插入新元素,并且保持新元素的格式不变。具体用法如图 2-7 所示。

```
In [8]:  language = ["python", "go", "java"]
         language.append("c")
         language.append("php")
         language

Out[8]:  ['python', 'go', 'java', 'c', 'php']

In [9]:  language = ["python", "go", "java"]
         language.append(["c", "php"])
         language

Out[9]:  ['python', 'go', 'java', ['c', 'php']]

In [10]: language = ["python", "go", "java"]
         language.extend(["c", "php"])
         language

Out[10]: ['python', 'go', 'java', 'c', 'php']
```

图 2-6　append 和 extend 用法示例

```
In [11]: language = ["python", "go", "java"]
         language.insert(1, "c")
         language

Out[11]: ['python', 'c', 'go', 'java']

In [12]: language = ["python", "go", "java"]
         language.insert(1, ["c", "php"])
         language

Out[12]: ['python', ['c', 'php'], 'go', 'java']
```

图 2-7　insert 用法示例

2.3.2　删(pop、remove、del)

列表中常用的删除方法有 3 种：第一种是从列表末尾删除元素，可使用 pop()方法；第二种是删除指定索引位置的元素，可使用 pop()，也可使用 del；第三种是删除列表中特定元素的值，可使用 remove()方法，如果列表中该元素值有多个，则删除第一个匹配的值，其他的不会删除。具体的用法如图 2-8 所示。

2.3.3　改、查

对于列表中的元素改写，先要找出元素的索引位置，然后再使用等号"＝"进行赋值，就可以改写列表中的元素值。具体示例如图 2-9 所示。

所谓的查找，实际上就是判断某一特定的元素是否存在于列表中。常用的查找

```
In [13]: language = ["python", "go", "java", "c"]
         language.pop()
         language
Out[13]: ['python', 'go', 'java']

In [14]: language = ["python", "go", "java", "c"]
         language.pop(1)
         language
Out[14]: ['python', 'java', 'c']

In [15]: language = ["python", "go", "java", "c"]
         del language[1]
         language
Out[15]: ['python', 'java', 'c']

In [16]: language = ["python", "go", "java", "c", "go"]
         language.remove("go")
         language
Out[16]: ['python', 'java', 'c', 'go']

In [17]: language = ["python", "go", "java", "c", "go"]
         language.remove("java")
         language
Out[17]: ['python', 'go', 'c', 'go']
```

图 2-8 pop()、remove()、del 用法示例

```
In [18]: language = ["python", "go", "java", "c"]
         language[1] = "php"
         language
Out[18]: ['python', 'php', 'java', 'c']
```

图 2-9 列表的改操作示例

方法有：in、not in、index 和 count。具体用法如表 2-8 所示。

表 2-8 列表的查操作用法示例

方法名及含义	例　　子
in 判断元素是否存在于列表中，存在则返回 True，不存在则返回 False	>>> language=["python", "go", "java", "c"] >>>"c" in language True >>>"php" in language False
not in 判断元素是否不存在于列表中，不存在则返回 True，存在则返回 False	>>> language=["python", "go", "java", "c"] >>>"c" not in language False >>>"php" not in language True

续表

方法名及含义	例　　子

index
与字符串的 index 用法相同，判断元素是否存在于列表中，存在则返回该元素在列表中的索引位置，不存在则直接报错
	>>> language=["python", "go", "java", "c"]
	>>> language.index("c")
	3
	>>> language.index("c", 1, 3)
	ValueError: 'c' is not in list

count
判断元素在列表中出现的次数，存在则返回次数，不存在则返回 0
	>>> language=["python", "go", "java", "c", "go"]
	>>> language.count("go")
	2
	>>> language.count("php")
	0

列表中的查询，直接使用索引即可访问该元素。要确保索引不要越界，并且列表的最后一个元素的索引是列表长度减去 1。

1. 使用索引查询元素

如果要取最后一个元素，除了计算索引位置外，还可以用 −1 做索引，直接获取最后一个元素，以此类推，可以获取倒数第 2 个、倒数第 3 个。具体操作如图 2-10 所示。

```
In [19]: language = ["python", "go", "java", "c"]
         language[1]
Out[19]: 'go'

In [20]: language = ["python", "go", "java", "c"]
         language[-1]
Out[20]: 'c'

In [21]: language = ["python", "go", "java", "c"]
         language[-2]
Out[21]: 'java'

In [22]: language = ["python", "go", "java", "c"]
         language[-3]
Out[22]: 'go'
```

图 2-10　使用索引查询列表元素

注意，列表 list 中的元素，可以是一个列表，这就是所谓的列表嵌套。在学习了字典之后，列表的元素也可以是字典。

```
>>> language = ["python", "go", ["php", "scala"], "java", "c"]
>>> len(language)
5
```

要注意 language 只有 5 个元素，其中 language[2]又是一个 list，如果拆开写就更容易理解了。因此，上述代码可改写为：

```
p = ["php", "scala"]
language = ["python", "go", p, "java", "c"]
```

要想取得"php"，可以写 p[0]或者 language[2][0]，因此 language 可以看成是一个二维数组，类似的还有三维、四维等数组，但在实际开发过程中，很少会用到三维及以上的数组。

2. 使用切片访问列表

这一点与字符串的访问相类似。其切片语法为：[起始：结束：步长]。选取的区间属于左闭右开型区间，即从“起始”位开始，到“结束”位的前一位结束(不包含结束位本身)。对应到数学中，就是前闭后开型区间，也可以理解为 start <=i < end。

例如：language=["python", "go", "php", "scala", "java", "c"]，则切片访问列表的示例如表 2-9 所示。

表 2-9　切片访问列表的示例

列表切片语法	含　义
language[0：3]=['python', 'go', 'php']	取下标 0～2 的元素
language[0：5]=['python', 'go', 'php', 'scala', 'java']	取下标 0～4 的元素
language[3：5]=['scala', 'java']	取下标 3、4 的元素
language[2：]=['php', 'scala', 'java', 'c']	取下标为 2 开始到最后的元素
language[1：−1]=['go', 'php', 'scala', 'java']	取下标为 1 开始到倒数第 2 个之间的元素
language[：：2]=['python', 'php', 'java']	从下标 0 开始到最后字符，每隔 2 个位置取出元素
language[5：1：−2]=['c', 'scala']	从下标 5 开始，到下标 2 逆序，每隔 2 个位置取出元素

2.3.4　列表的循环遍历

如果要查看列表的所有元素，则需要对列表进行循环遍历。以下是两种常见的循环语句写法。

（1）使用 for 循环

```
languages = ["python", "go", "php", "scala", "java", "c"]
for language in languages:
    print(language)
```

（2）使用 while 循环

```
languages = ["python", "go", "php", "scala", "java", "c"]
length = len(languages)
i = 0
while i < length:
    print(languages[i])
    i += 1
```

2.3.5　排序（sort，reverse）

sort 方法是将 list 按特定顺序重新排列，排序方式默认为由小到大，参数 reverse=True，可改为倒序，由大到小排序。

reverse 方法可以直接将 list 中所有的元素进行逆置。

```
>>> a = [1, 4, 2, 3]
>>> a.reverse( )
>>> a
[3, 2, 4, 1]
>>> a.sort( )
>>> a
[1, 2, 3, 4]
>>> a.sort(reverse = True)
>>> a
[4, 3, 2, 1]
```

2.3.6　列表的其他操作符

列表对＋和×的操作符与字符串相似，也可以进行成员检查。＋号用于组合列

表,×号用于重复列表。其具体用法如表 2-10 所示。

表 2-10　+、×操作示例

表　达　式	输 出 结 果	描　　述
[1,2,3]+[4,5,6]	[1,2,3,4,5,6]	将列表拼接组合起来
["python"]× 3	["python","python","python"]	重复
1 in [1,2,3]	True	判断元素是否存在于列表中
for i in [1,2,3]: print(i)	1 2 3	迭代

2.4　集合 set

在学数学的时候,都会学习到集合的概念。在数学中,集合实际上就是"确定的一堆东西"。集合里的"东西",叫作元素。由一个或多个确定的元素所构成的整体叫作集合。若 x 是集合 A 的元素,则记作 $x \in A$。

集合中的元素有 3 个特征。

(1) 确定性(集合中的元素必须是确定的)。

(2) 互异性(集合中的元素互不相同)。例如:集合 $A=\{1,a\}$,则 a 不能等于 1。

(3) 无序性(集合中的元素没有先后之分),如集合{3, 4, 5}和{3, 5, 4}算作同一个集合。

在 Python 中,集合也是类似的,用 set()表示,也可以使用大括号{ }来表示集合,其中的元素也是无顺序的。由于在实际编写代码的过程中,集合的用处比较少,下面简单介绍集合的常见用法。

【注意】

由于集合具有互异性(集合中的元素互不相同),因此可以对一个有重复元素的列表,添加上 set()之后,就会将重复的元素隐藏。

2.4.1　创建集合

创建集合时,使用 set()即可。如 var = set([1, 3, 5, 6, 4, 2]),如图 2-11 所示。

```
In [23]: var = set([1,3,5,6,4,2])
         var
Out[23]: {1, 2, 3, 4, 5, 6}

In [24]: for i in var:
             print(i)

         1
         2
         3
         4
         5
         6
```

图 2-11　创建集合示例

由图 2-11 可以看出,集合内部实际上已经对元素进行了排序。那么,集合中能不能包含不同类型的元素呢?具体如图 2-12、图 2-13 和图 2-14 所示。

```
In [25]: var = {1,"d",3,"c",5,"a",6,"f",4,"b",2}
         var
Out[25]: {1, 2, 3, 'f', 5, 6, 4, 'a', 'b', 'c', 'd'}

In [26]: for i in var:
             print(i)

         1
         2
         3
         f
         5
         6
         4
         a
         b
         c
         d
```

图 2-12　集合中包含整型和字符型示例

```
In [27]: var = {1,3,5,6,4,2,{12}}
         var
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-f5912640a9c1> in <module>()
----> 1 var = {1,3,5,6,4,2,{12}}
      2 var

TypeError: unhashable type: 'set'
```

图 2-13　集合中包含整型和字典示例

由此可以看出,如果放入的元素类型是集合或者列表,则会报 TypeError 错误。如果集合里都是整型和字符型的元素,则循环遍历集合的输出结果就不会进行排序。那么是不是说,如果元素类型相同,就一定会排序呢?具体如图 2-15 所示。

```
In [28]: var = {1,3,5,6,4,2,[12, 34]}
         var

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-5b774cfbc444> in <module>()
----> 1 var = {1,3,5,6,4,2,[12, 34]}
      2 var

TypeError: unhashable type: 'list'
```

图 2-14 集合中包含整型和列表示例

```
In [29]: var = {"d","a","f","c","e","d"}
         var

Out[29]: {'a', 'c', 'd', 'e', 'f'}

In [30]: for i in var:
             print(i)

e
f
a
c
d
```

图 2-15 集合中字符型排序示例

可见,如果集合的元素全是字符型,那么实际上输出的结果并没有进行排序。

总之,集合中可以放不同类型的元素,但是存放的元素只能为数值型和字符型,不能是列表类型和集合类型。如果元素都是数值型,则循环变量的输出结果为从小到大排序后的结果;如果元素都是字符型,循环变量的输出结果并不会进行排序;如果元素既有数值型又有字符型,则循环变量的输出结果也不会进行排序。

2.4.2 集合的增、删

如果向集合中添加元素,则使用 add()函数。集合的 add()示例如图 2-16 所示。

```
In [31]: names = {"Jack","David","Rose"}
         names.add("Jone")
         print(names)

{'David', 'Jone', 'Rose', 'Jack'}
```

图 2-16 集合的 add()示例

如果要随机删除集合中的一个元素，可以使用 pop()方法；如果要删除指定的元素，可以使用 remove()和 discard()方法。但当 remove()方法找不到指定的元素时，会报错；而当 discard()方法找不到指定的元素时，并不会报错。如图 2-17、图 2-18 和图 2-19 所示。

```
In [32]: names = {"David", "Jack", "Jone", "Rose"}
         result = names.pop()
         print(result)

David
```

图 2-17　集合的 pop()示例

```
In [33]: names = {"David", "Jack", "Jone", "Rose"}
         names.remove("Jack")
         names

Out[33]: {'David', 'Jone', 'Rose'}

In [34]: names = {"David", "Jack", "Jone", "Rose"}
         names.remove("Jack1")
         names

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-34-16d1bfb1ed44> in <module>()
      1 names = {"David", "Jack", "Jone", "Rose"}
----> 2 names.remove("Jack1")
      3 names

KeyError: 'Jack1'
```

图 2-18　集合的 remove()示例

```
In [35]: names = {"David", "Jack", "Jone", "Rose"}
         names.discard("Jack")
         names

Out[35]: {'David', 'Jone', 'Rose'}

In [36]: names = {"David", "Jack", "Jone", "Rose"}
         names.discard("Jack1")
         names

Out[36]: {'David', 'Jack', 'Jone', 'Rose'}
```

图 2-19　集合的 discard()示例

2.4.3　集合的交、并、补等操作

在数学中，集合有交集、并集、补集等概念，那么 Python 中是否支持交集、并集、补集等操作呢？集合的交、并、补等用法如表 2-11 所示。集合的交、并、补操

作示例如图 2-20 所示。

表 2-11 集合的交并补等用法

数学符号	Python 符号和内置方法	含 义
－或\	－或 difference()	差集
∩	& 或 intersection()	交集
∪	|或 union()	并集、合集
≠	!=	不等于
=	==	等于
∈	in	是成员关系
∉	not in	不是成员关系

```
In [37]: names1 = {"David", "Jack", "Jone", "Rose"}
         names2 = {"Tom", "Lily", "Rose"}
         names1.difference(names2)
Out[37]: {'David', 'Jack', 'Jone'}

In [38]: names1 = {"David", "Jack", "Jone", "Rose"}
         names2 = {"Tom", "Lily", "Rose"}
         names1.union(names2)
Out[38]: {'David', 'Jack', 'Jone', 'Lily', 'Rose', 'Tom'}

In [39]: names1 = {"David", "Jack", "Jone", "Rose"}
         names2 = {"Tom", "Lily", "Rose"}
         names1.intersection(names2)
Out[39]: {'Rose'}
```

图 2-20 集合的交并补操作示例

2.5 字典 dictionary

在 Python 中，字典用大括号{ }表示，但字典与集合不同，字典的{ }里面必须有键和对应的值，而集合的{ }里只需要有值即可。字典的每个元素是由两部分组成，即键和值，俗称键值对(key-value)。每个键值对 key=> value 用冒号"："分割，每个键值对之间用逗号","分割。其语法格式如下：

```
d = {key1 : value1, key2 : value2 ,……}
```

【注意】

(1) 字典的键一般是唯一的，如果字典的键发生重复，则最后一个键值对就会替

换掉前面的键值对。字典含有重复键的输出示例如图 2-21 所示。

```
In [40]: dict = {"a": 1, "b": 2, "b": '3'}
         dict["b"]

Out[40]: '3'

In [41]: dict

Out[41]: {'a': 1, 'b': '3'}
```

图 2-21　字典含有重复键的输出示例

(2) 字典的值可以取任何数据类型,但是键必须是不可变类型,如数字、字符串和元组,但是不能使用列表作为字典的键。字典键是列表时的示例如图 2-22 所示。

```
In [42]: dict = {["name"]: "Tom", "b": 2, "b": '3'}
         dict

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-42-afc6b08f35b6> in <module>()
         ----> 1 dict = {["name"]: "Tom", "b": 2, "b": '3'}
               2 dict

         TypeError: unhashable type: 'list'
```

图 2-22　字典键是列表时的示例

2.5.1　字典的查找操作

在字典中,如果想获取一个元素,那么只需要根据字典的键,就可以访问到对应的值。例如:在学生信息系统中,可以查询学生姓名,也可以查询学生的学号。字典的查找操作示例如图 2-23 所示。

```
In [43]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
         print(stu_info["name"])
         print(stu_info["id"])

         Tom
         100
```

图 2-23　字典的查找操作示例

在这个例子中,name、id 都是字典的键,它们存储了对应的姓名和学号。当然,也可以查询性别、地址等信息。

如果查询的键不存在,比如,想查询 stu_info 中的年龄 age 和对应的数值,得到的结果是什么呢?字典中不存在对应键的操作示例如图 2-24 所示。

在查询年龄 age 时,stu_info 中并没有存储 age 和对应的值,导致结果报错。错

```
In [44]: print(stu_info["age"])

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-44-3c4e1ca6ebeb> in <module>()
----> 1 print(stu_info["age"])

KeyError: 'age'
```

图 2-24　字典中不存在对应键的操作示例

误信息为：KeyError：'age'，即提示 stu_info 中并没有 age 这个键。当不确定某一个字典中是否存在某个键，但是又想获取对应的值时，可以使用 get()方法，同时还可以设置获取值的默认值。字典的 get()方法示例如图 2-25 所示。

```
In [45]: print(stu_info.get("age"))

None

In [46]: print(stu_info.get("age",15))

15
```

图 2-25　字典的 get()方法示例

如果 stu_info 中不存在 age 这个键，则 get()方法返回 None。给 get()方法传入一个默认值 15。当不存在这个键时，则返回默认值 15。

2.5.2　字典的增、改操作

在查找出元素的基础上，就可以对元素进行添加或修改。当查找的键存在时，使用变量名['key'] = value 的方式，实际上是对元素进行修改操作。当查找的键不存在时，使用变量名['key'] = value 的方式，可以将该 key 和对应的 value 添加到字典中。字典的增操作示例如图 2-26 所示。

```
In [47]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
         print(stu_info)
         stu_info["age"] = 18
         print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China', 'age': 18}
```

图 2-26　字典的增操作示例

字典的每个元素中的数据是可以修改的，只要通过 key 找到，就可以修改。字典的改操作示例如图 2-27 所示。

```
In [48]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
         print(stu_info)
         stu_info["address"] = "England"
         print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'England'}
```

图 2-27　字典的改操作示例

2.5.3　字典的删操作

如果想要删除字典中的元素，可以使用 del、pop()或 clear()。del 用于删除指定的元素，也可用于删除整个字典，pop()用于删除指定元素，clear()用于删除整个字典。字典的 del 操作示例如图 2-28 所示。

```
In [49]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
         print(stu_info)
         del stu_info["sex"]
         print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
{'name': 'Tom', 'id': 100, 'address': 'China'}
```

图 2-28　字典的 del 操作示例

del 除了可以删除指定的元素之外，还可以删除整个字典。del 删除整个字典操作示例如图 2-29 所示。

```
In [50]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
         print(stu_info)
         del stu_info
         print(stu_info)

{'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-50-68e2b1254419> in <module>()
      2 print(stu_info)
      3 del stu_info
----> 4 print(stu_info)

NameError: name 'stu_info' is not defined
```

图 2-29　del 删除整个字典操作示例

pop()用于删除指定元素，如果键不存在，则报错 KeyError。字典的 pop()操作示例如图 2-30 所示。

clear()用于删除整个字典，与 del 不同，clear()只是删除字典里的键值对，得到的

```
In [51]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
         print(stu_info)
         stu_info.pop("sex")
         print(stu_info)

         {'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
         {'name': 'Tom', 'id': 100, 'address': 'China'}

In [52]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
         print(stu_info)
         stu_info.pop("age")
         print(stu_info)

         {'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}

         ---------------------------------------------------------------------------
         KeyError                                  Traceback (most recent call last)
         <ipython-input-52-fb853e565135> in <module>()
               1 stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
               2 print(stu_info)
         ----> 3 stu_info.pop("age")
               4 print(stu_info)

         KeyError: 'age'
```

图 2-30　字典的 pop()操作示例

结果是一个空字典，del 则是删除了整个字典结构。可以类比于一本书，clear()是把书里的所有内容删除，虽然变成了空白页，但是书的结构还在；del 则是完全销毁了这本书，即书的结构也就不存在了。字典的 del 和 clear()对比示例如图 2-31 所示。

```
In [53]: stu_info = {"name": "Tom", "id": 100, "sex": "male", "address": "China"}
         print(stu_info)
         stu_info.clear()
         print(stu_info)

         {'name': 'Tom', 'id': 100, 'sex': 'male', 'address': 'China'}
         {}
```

图 2-31　字典的 del 和 clear()对比示例

2.5.4　字典的常用方法

除了上述介绍的字典的增删改查等操作，字典还内嵌了一些其他的方法，字典的内嵌方法示例具体如表 2-12 所示。

表 2-12　字典的内嵌方法示例

方法名及含义	例　　子

len()
返回字典中键值对的个数
	>>> stu_info = {'name': 'Tom', 'id': 100}
	>>> len(stu_info)
	2

续表

方法名及含义	例　　子

keys()
返回字典中所有 key 的列表
	>>> stu_info = {'name': 'Tom', 'id': 100}
	>>> stu_info. keys()
	['name', 'id']

values()
返回字典所有 value 的列表
	>>> stu_info = {'name': 'Tom', 'id': 100}
	>>> stu_info. values()
	['Tom', 100]

items()
返回一个列表，列表里的元素是(键，值)元组
	>>> stu_info = {'name': 'Tom', 'id': 100}
	>>> stu_info. items()
	[('name', 'Tom'), ('id', 100)]

has_key(key)
如果字典的键中含有指定的 key，返回 True，否则返回 False。(此方法在 Python3 中已被删除)
	>>> stu_info = {'name': 'Tom', 'id': 100}
	>>> stu_info. has_key('name')
	True
	>>> stu_info. has_key('age')
	False

in 如果字典的键中含有指定的 key，返回 True，否则返回 False。(Python2、3 都可以使用)	>>> stu_info = {'name': 'Tom', 'id': 100} >>>'name' in stu_info True >>>'age' in stu_info False

2.5.5　有序字典

如果希望对字典做迭代，按照元素初始添加的顺序进行结果输出，那么就可以使用 collections 包下的 OrderedDict 方法。

对字典做迭代时，它会严格按照元素初始添加的顺序进行。有序字典 OrderedDict 的遍历示例如图 2-32 所示。

```
In [54]: from collections import OrderedDict
         od = OrderedDict()
         od["first"] = 1
         od["second"] = 2
         od["third"] = 3
         for key,value in od.items():
             print(key, value)

         first 1
         second 2
         third 3
```

图 2-32　有序字典 OrderedDict 的遍历示例

通过查看源码,如图 2-33 所示,OrderedDict 内部维护了一个双向链表,它会根据元素加入的顺序来排列键的位置。第一个新加入的元素被放置在链表的末尾,再对已存在的键做重新赋值,不会改变键的顺序。

```
class OrderedDict(dict):
    'Dictionary that remembers insertion order'
    # An inherited dict maps keys to values.
    # The inherited dict provides __getitem__, __len__, __contains__, and get.
    # The remaining methods are order-aware.
    # Big-O running times for all methods are the same as regular dictionaries.

    # The internal self.__map dict maps keys to links in a doubly linked list.
    # The circular doubly linked list starts and ends with a sentinel element.
    # The sentinel element never gets deleted (this simplifies the algorithm).
    # The sentinel is in self.__hardroot with a weakref proxy in self.__root.
    # The prev links are weakref proxies (to prevent circular references).
    # Individual links are kept alive by the hard reference in self.__map.
    # Those hard references disappear when a key is deleted from an OrderedDict.

```

图 2-33 OrderedDict 的简介说明

由于采用了双向链表,OrderedDict 的大小是普通字典的两倍多,因此,在想要构建一个涉及大量 OrderedDict 实例的数据结构时,首先需要判断增加的内存开销是否划算,然后再决定是否使用 OrderedDict。

在实际开发过程中只知道常用的操作还是不够的,接到一个需求之后,最主要的任务就是编写出实现具体功能的函数。

2.6 函数

在数学中,给出长方形的长度 a 和宽度 b,求出长方形的面积 $S=ab$。在 Python 中,输出的结果实际上就相当于 return 语句,长度和宽度实际上就相当于函数的输入参数。

在 Python 中,定义函数的语法如下所示:

```
def 函数名(输入参数 1,输入参数 2,……):
    函数体
    return 返回结果
```

比如这个计算长方形面积的函数,就可以写成:

```
def cal_area(length, width):
    s = length * width
    return s
```

在定义函数时，小括号中的参数，用于接收参数，称之为“形参”。在调用函数时，括号中的参数，传递给函数的参数，称之为“实参”。

在实际开发过程中，常常需要考虑，该函数需不需要参数？如果需要，需要几个参数？该函数需不需要返回值？需要返回几个返回值？

根据这些，可以把函数分为 4 种类型，如表 2-13 所示。

表 2-13 函数的 4 种类型

函数类型	举例
① 无参数，无返回值的函数 此类函数，不能接收参数，也没有返回值。一般情况下，打印提示类的功能，可使用这类函数	def printHello(): print('欢迎光临本店') print('祝您用餐愉快')
② 无参数，有返回值的函数 此类函数，不能接收参数，但是可以返回某个数据。一般情况下，如采集数据，可使用这类函数	def getTemperature(): return 24
③ 有参数，无返回值的函数 此类函数，能接收参数，但不返回数据。一般情况下，该函数主要用于对变量进行赋值，如最常见的 set()函数	def setData(num): self.num = num

④ 有参数，有返回值的函数
此类函数，不仅能接收参数，还可返回某个数据。一般情况下，对于数据处理并需要结果的应用，可用这类函数
	def calculateNum(num):
	    result = 0
	    i = 1
	    while i<=num:
	        result = result + i
	        i+=1
	    return result

在实际开发过程中，第一类函数主要用于打印代码调试、警告、错误信息等，方便开发人员进行错误跟踪，及时对代码进行修改和优化；第二类函数主要用于获取数据，最常见的函数就是 get 方法；第三类函数主要用于设置数据，最常见的函数就是 set 方法；第四类函数是重点，在处理数据的过程中，需要知道传进来的参数是什么，对其进行处理后，要将处理后的结果再返回出去，以便后续的操作。

2.7 小结

本章主要介绍了常见的 Python 数据类型的用法以及函数的 4 种类型。在实际开发过程中,可以通过查看源码,来帮助理解底层的实现方式。遇到问题的时候,多思考,多动手尝试。在此基础上,明确代码的处理逻辑和顺序,但不重复发明轮子,使用已经开发好的包或者第三方库,可以达到事半功倍的目的,从而提高工作效率。

第 3 章

Python下的实际应用

Python 是一种代表简单主义思想的语言，有“胶水语言”之称。目前，Python 的应用层面也十分广泛，主要包括数据库的连接、系统编程、网络爬虫、人工智能、Web 开发、系统运维、大数据、云计算、图形界面等。在当前环境下，Python 重点应用于机器学习和人工智能方面。

3.1 Python 连接 MySQL 数据库



视频讲解

MySQL 数据库是企业最常用的数据库之一，而 MySQL 数据库最常用的功能是 select 查询。数据分析人员希望可以用 Python 直接连接 MySQL 数据库，然后读取数据到 pandas 框架中。

首先安装好 MySQL 数据库，安装步骤详见附录 B。然后安装 pymysql，安装语句为：pip install pymysql，安装好这个包以后，在 Jupyter Notebook 中使用以下代码：

```
import pandas as pd
import pymysql
conn = pymysql.connect(
    host = '主机 IP',
```

```
    user = '用户名',
    password = '密码',
    db = '数据库名',
    port = 端口,
    charset = 'utf8')
table = "select...from..."
```

上面的 table 就是连接 mysql 后查询到的表。然后用下面代码,就可以直接把数据读取到 pandas 框架中了。数据读取到 pandas 框架后,就可以使用 pandas 的常见功能对数据进行处理,比如填补缺失值、删除数据等。

```
data = pd.read_sql(table, conn)
```

3.2 Python 连接 MongoDB 数据库



视频讲解

MongoDB 数据库也是企业常用的数据库之一。MongoDB 数据库是一个介于关系数据库和非关系数据库之间的产品,是非关系数据库中功能最丰富、最像关系数据库的一种数据库。

相对于 MySQL 数据库,MongoDB 数据库具有独特的优势。同样地,Python 也可以直接连接 MongoDB 数据库,并把数据读取到 pandas 框架中。安装步骤详见附录 C。

首先安装 pymongo,安装语句为:pip install pymongo。安装步骤详见附录 F。安装好 pymongo 后,使用方法如下:

```
import pandas as pd
from pymongo import MongoClient
client = MongoClient('主机 IP', port = 端口)
db = client.数据库名
db.authenticate('用户名', '密码')
table = db.表名
```

与 MySQL 数据库一样,上面的 table 就是连接 MongoDB 后查询到的表,然后使用下面代码,就可以直接把数据读取到 pandas 框架了,其中,find 括号里可以加入 MongoDB 数据库查询语句。

```
data = pd.DataFrame(list(table.find()))
```

3.3 结巴分词和词云图



视频讲解

Python可以处理文本数据，常见的有NLTK自然语言处理工具包、LSTM长短期记忆网络，但这些工具包基本上都用于处理英文。这里介绍下简单且常用的中文语言处理工具包jieba。

下面将以Mac系统为例讲解jieba和wordcloud的安装方法，安装语句为：pip install jieba，pip install wordcloud。步骤参考附录F第三方包安装步骤。

本节以网络下载的小说《天龙八部》为例，讲解jieba和word cloud的用法。

```
import pandas as pd
import os
os.chdir('../data')
character = open('天龙八部人物表.txt', encoding = 'utf-8').read()
'''添加词性'''
import re
data = re.split(r'\s+|: |,', character)
data = pd.DataFrame(data, columns = ['姓名'])
data['词性'] = 'nr'
data.to_excel('天龙八部人物分词.xlsx', index = False, header = None)
'''添加自定义词库'''
import jieba
jieba.enable_parallel(4)
jieba.load_userdict('天龙八部人物词典.txt')
'''加载停用词库'''
stopwords = [line.rstrip() for line in open('停用词表.txt', encoding = 'utf-8')]
'''jieba分词'''
def seg_sentence(sentence):
    sentence_seged = jieba.cut(sentence.strip())
    outstr = ''
    for word in sentence_seged:
        if word not in stopwords:
            if word != '\t':
                outstr += word
                outstr += ' '
    return outstr
inputs = open('天龙八部.txt', 'r', encoding = 'GB18030')
outputs = open('天龙八部分词.txt', 'w', encoding = 'utf-8')
```

```
for line in inputs:
    line_seg = seg_sentence(line)
    outputs.write(line_seg + '\n')
outputs.close()
inputs.close()
```

分词过后，就可以进行文本分析了。先加载分词后的 txt 文本：

```
text = open('天龙八部分词.txt', encoding = 'utf-8').read()
```

然后指定词性，提取关键词，并打印出由 TF-IDF 方法计算出的权重最大的前 10 人物，效果如图 3-1 所示。

```
'''指定词性, 提取关键词: TF-IDF'''
import jieba.analyse

n=100            #指定关键词数量

result=jieba.analyse.extract_tags(text, topK=n, withWeight=True, allowPOS=('nr',))

result[:10]      #打印前10个最重要的人物
```

```
[('段誉', 0.5891293630142107),
 ('萧峰', 0.4638848540025745),
 ('乔峰', 0.3369411093846939),
 ('慕容复', 0.2771830168385107),
 ('王语嫣', 0.26712617741403344),
 ('武功', 0.23669144784504287),
 ('段正淳', 0.22647654172059356),
 ('阿朱', 0.21979838728524273),
 ('木婉清', 0.21805626004123815),
 ('鸠摩智', 0.1742127244004566)]
```

图 3-1 TF-IDF 计算出的权重最大的前 10 人

```
'''词云关键词'''
keywords = dict()
for i in result:
    keywords[i[0]] = i[1]
'''设置词云属性'''
from PIL import Image
import numpy as np
from wordcloud import WordCloud, ImageColorGenerator
import matplotlib.pyplot as plt
image = Image.open('图片.jpeg')
graph = np.array(image)
wc = WordCloud(font_path = '/Library/Fonts/Songti.ttc',    # 设置字体
               background_color = 'White',                 # 设置背景颜色
               max_words = n,                              # 设置词云显示的最大词数
               mask = graph)                               # 设置背景样式
```

最后效果如图 3-2 所示。

```
In [41]: '''生成词云'''
         wc.generate_from_frequencies(keywords)
         plt.imshow(wc)
         image_color = ImageColorGenerator(graph)
         plt.imshow(wc.recolor(color_func=image_color))
         plt.axis("off")
         plt.show()
         wc.to_file('词云.jpg')
```



图 3-2 词云生成图

3.4 简单社交网络



视频讲解

目前,社交网络是大数据时代的热门课题之一,正式的社交网络图谱一般用 Neo4j 来做,Neo4j 的入门与应用将在本书第 8 章中进行详细介绍。本章将介绍 Python 语言中的简单社交网络。

```
import pandas as pd
data1 = pd.read_excel('牛魔王关系图.xlsx')
data2 = pd.read_excel('孙悟空关系图.xlsx')
data3 = pd.read_excel('唐僧关系图.xlsx')
data4 = pd.read_excel('猪八戒关系图.xlsx')
data = pd.concat([data1, data2, data3, data4])
```

简单社交网络数据展示图,如图 3-3 所示。

Source 是源节点,Target 是目标节点,Weight 是关系权重。

在社交网络中有 4 个重要的定义。

(1) 度中心性:一个节点在网络中的连接数。

```
In [5]: data.head()
```

Out[5]:

	Source	Target	Weight
0	牛魔王	铁扇公主	7
1	牛魔王	红孩儿	7
2	牛魔王	孙悟空	5
3	牛魔王	玉面狐狸	6
4	铁扇公主	红孩儿	8

图 3-3 简单社交网络数据展示图

(2) 接近中心性：识别集群内部被高度连接的节点。

(3) 中介中心性：识别连接不同集群的节点。

(4) 网页排名：衡量节点活跃度，节点越活跃，pangrank 值越大。

```
import networkx as nx
graph = nx.from_pandas_dataframe(data, 'Source', 'Target', edge_attr = ['Weight'])
```

度中心性、接近中心性、中介中心性、网页排名分别如图 3-4、图 3-5、图 3-6 和图 3-7 所示。

```
'''度中心性：一个节点在网络中的连接数'''
degree_df = pd.DataFrame(columns=['度中心性'])
degree_df['度中心性'] = pd.Series(nx.degree_centrality(graph))
degree_df.sort_values('度中心性', ascending=False)[:10].plot(kind='barh')
plt.show()
```

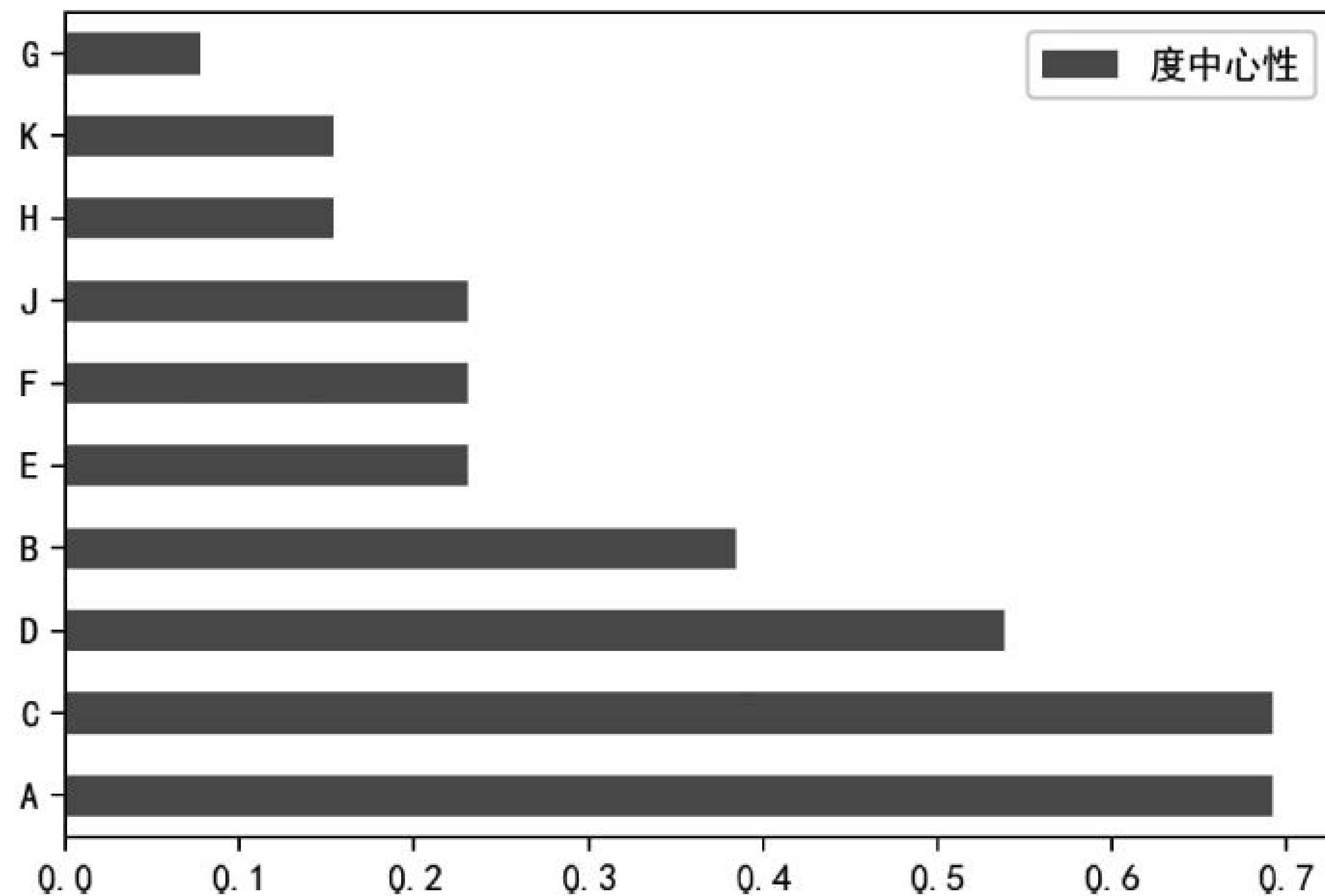


图 3-4 度中心性展示图

```
'''接近中心性：具有高接近中心性的节点通常在集群之间被高度连接'''
closeness_df = pd.DataFrame(columns=['接近中心性'])
closeness_df['接近中心性'] = pd.Series(nx.closeness_centrality(graph))
closeness_df.sort_values('接近中心性', ascending=False)[:10].plot(kind='barh')
plt.show()
```

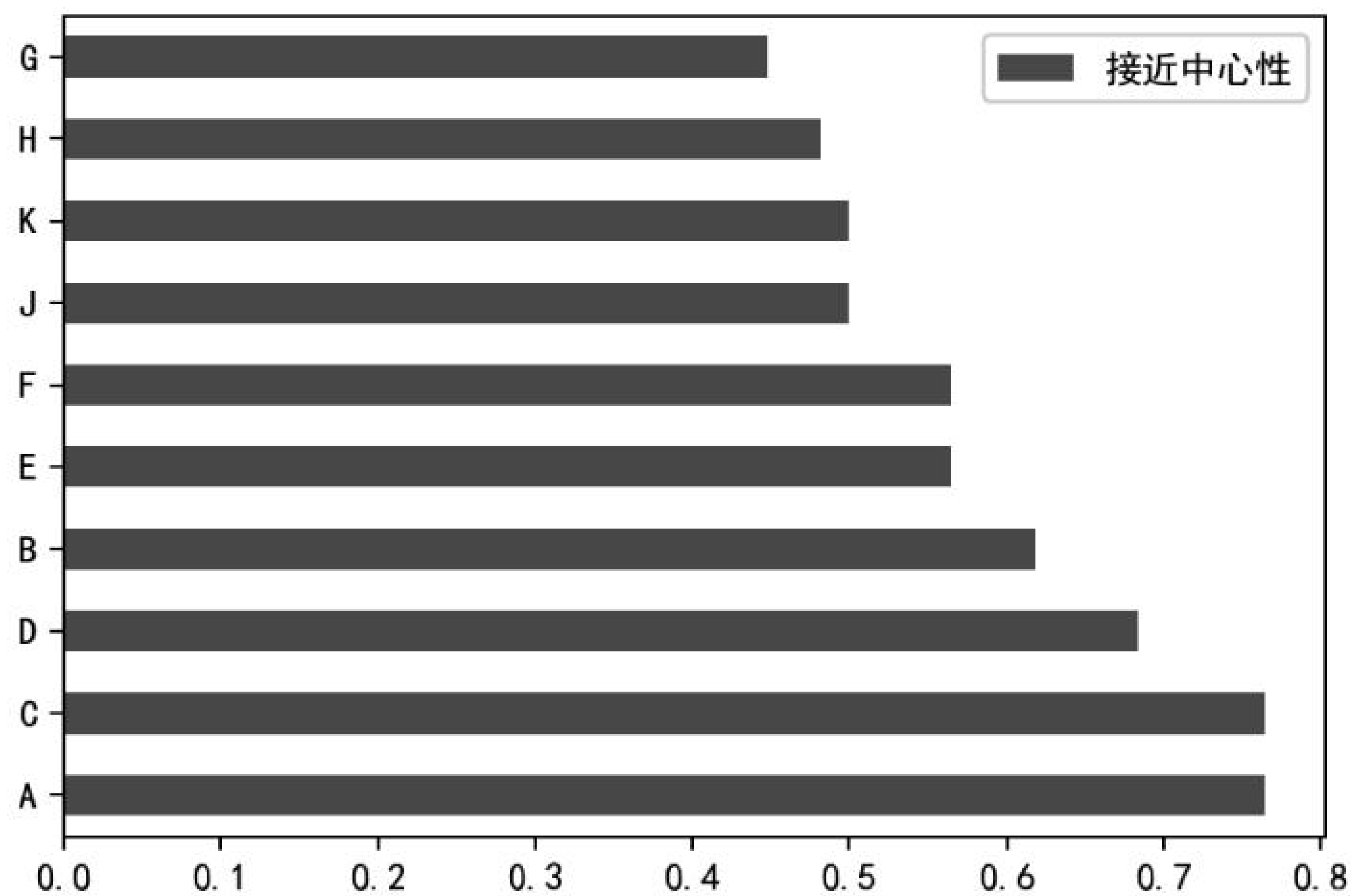


图 3-5　接近中心性展示图

```
'''中介中心性：识别连接不同集群的节点'''
betweenness_df = pd.DataFrame(columns=['中介中心性'])
betweenness_df['中介中心性'] = pd.Series(nx.betweenness_centrality(graph))
betweenness_df.fillna(0, inplace=True)
betweenness_df.sort_values('中介中心性', ascending=False)[:10].plot(kind='barh')
plt.show()
```

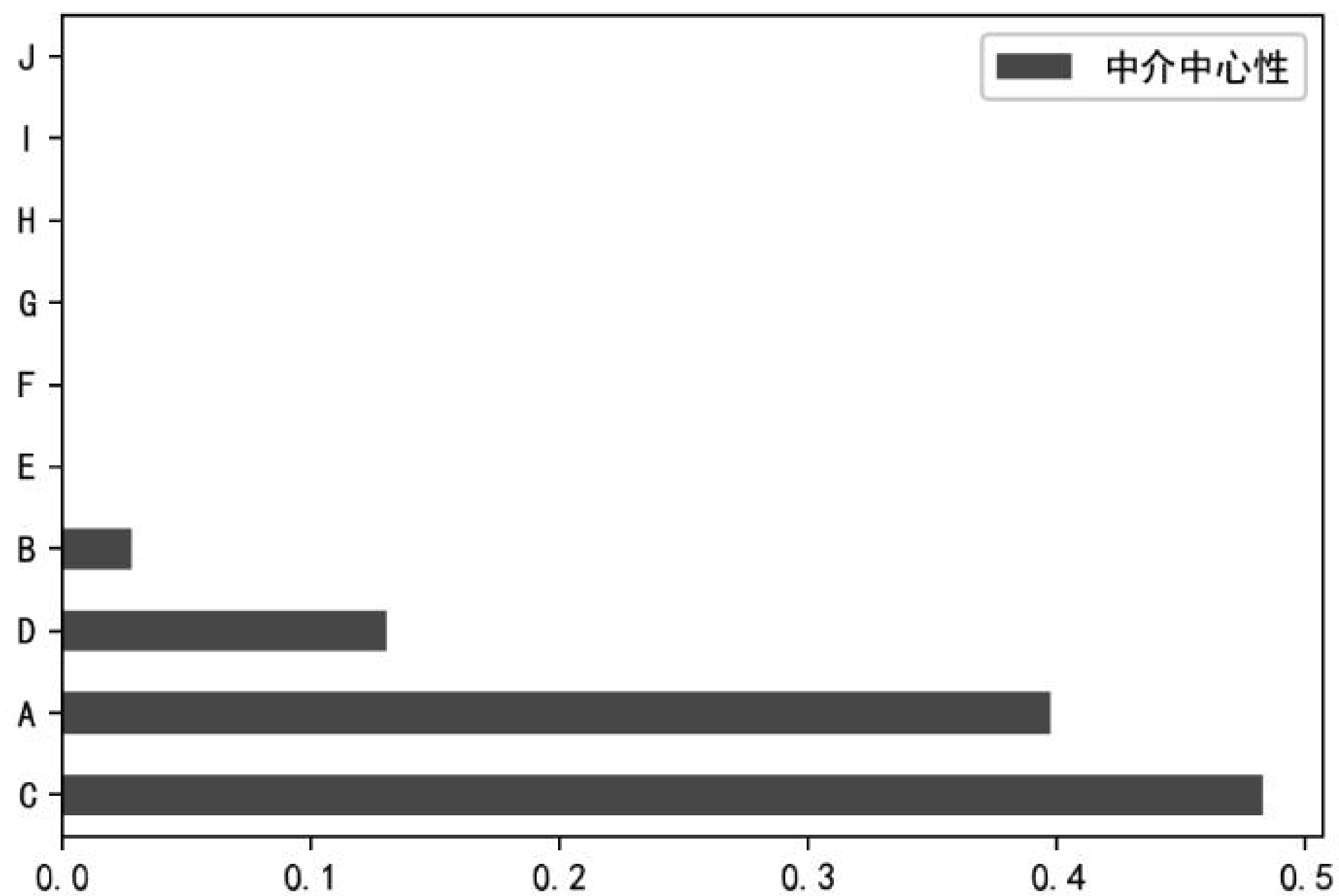


图 3-6　中介中心性展示图

```
'''网页排名：找出活跃用户'''
pagerank_df = pd.DataFrame(columns=['网页排名'])
pagerank_df['网页排名'] = pd.Series(nx.pagerank(graph))
pagerank_df.sort_values('网页排名', ascending=False)[:10].plot(kind='barh')
plt.show()
```

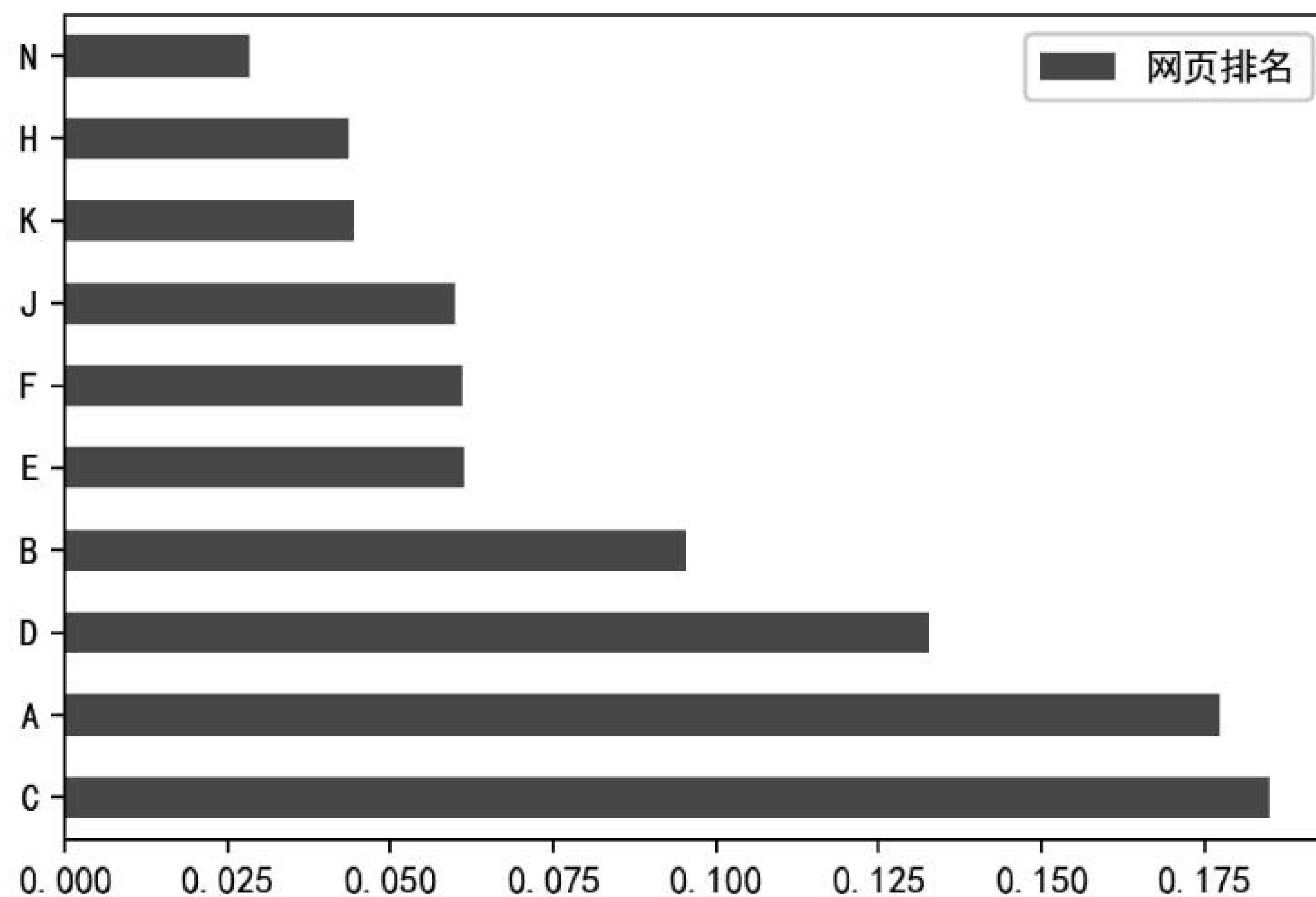


图 3-7 网页排名展示图

社交网络可视化代码如下：

```
plt.figure(figsize = (20, 10))
node_color = [graph.degree(v) for v in graph]
# 节点颜色由节点的度决定
node_size = [5000 * nx.degree_centrality(graph)[v] for v in graph]
# 节点的大小由 degree centrality 决定
edge_width = [0.2 * graph[u][v]['Weight'] for u, v in graph.edges()]
# 边的宽度由权重决定
pos = nx.spring_layout(graph)
nx.draw_networkx(graph, pos, node_size = node_size, node_color = node_color, alpha = 0.7,
with_labels = False, width = edge_width)
top = degree_df.sort_values('度中心性', ascending = False)[: 10]
# 最重要的 top10 人物
top = top.index.values.tolist()
labels = {role: role for role in top}
# 创建 label,不同的度量方法.标签不一样
nx.draw_networkx_labels(graph, pos, labels = labels, font_size = 20)
# 添加 label
plt.show()
nx.write_gexf(graph, '度中心性.gexf')
```

社交网络可视化结果，如图 3-8 所示。

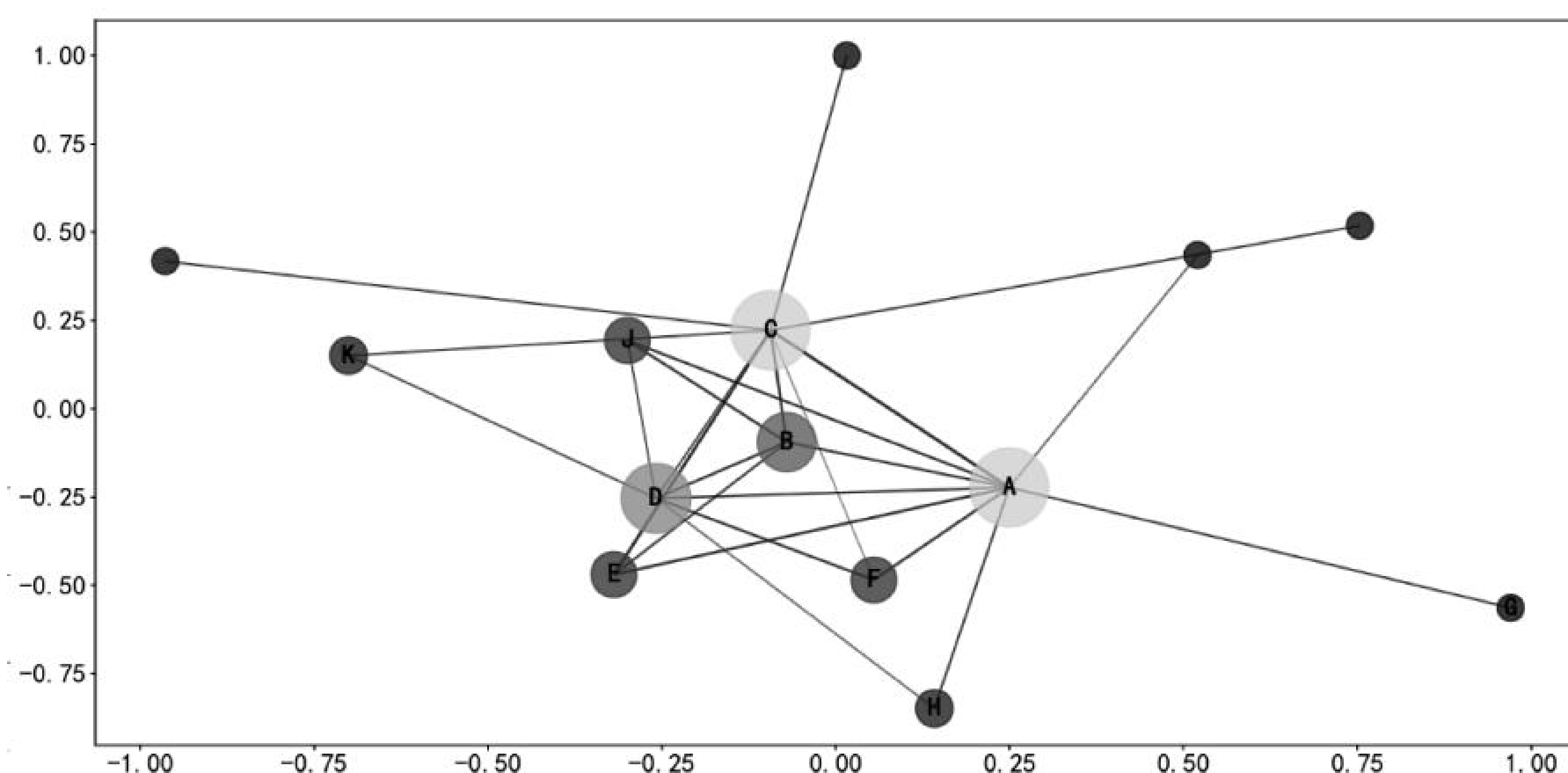


图 3-8 社交网络可视化结果展示图

最小社区探测代码如下：

```
k = 3
klist = list(nx.k_clique_communities(graph,k))
print ("生成的社区数: %d" % len(klist))
plt.figure(figsize = (20, 10))
nx.draw_networkx(graph, pos = pos, nodelist = klist[0], node_size = node_size, node_color
 = 'r', alpha = 0.7, with_labels = False, width = edge_width)
nx.draw_networkx(graph, pos = pos, nodelist = klist[1], node_size = node_size, node_color
 = 'y', alpha = 0.7, with_labels = False, width = edge_width)
nx.draw_networkx_labels(graph, pos, labels = labels, font_size = 20)
plt.show()
```

最小社区探测可视化结果，如图 3-9 所示。

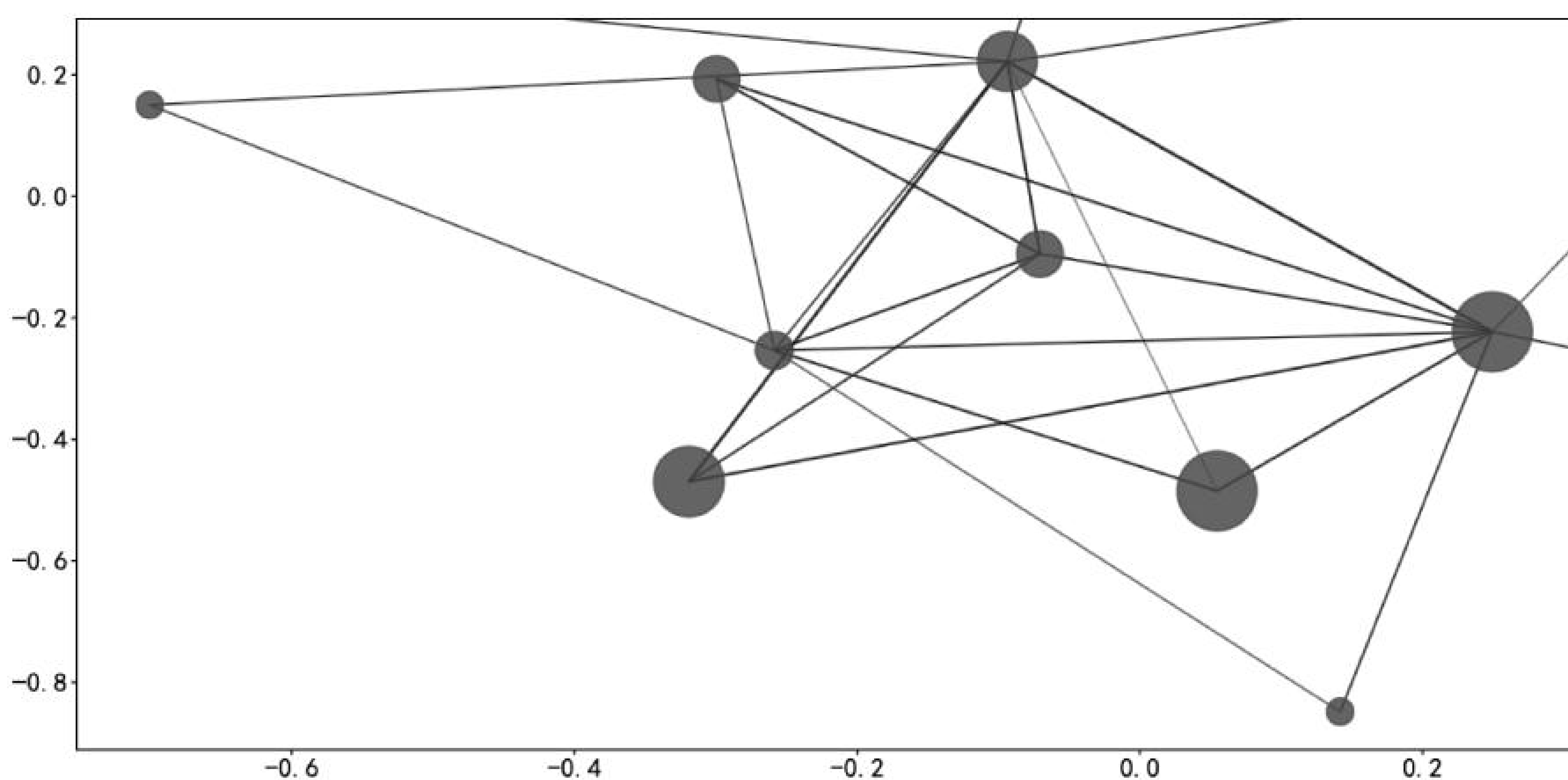


图 3-9 最小社区探测可视化展示图

上述代码指定 $k=3$，即一个社区至少有 3 个节点两两相连，而这里生成了两个社区。在金融领域，社区探测法常用于寻找欺诈团体。

3.5 JSON 解析



视频讲解

在进行数据分析时，从数据库中导出的数据，可能是 JSON 格式的。那么 JSON 是什么呢？JSON 是 JavaScript Object Notation 的缩写，是一种轻量级的数据交换格式，或者说是一个像字典一样的字符串。下面以读取 1 个 JSON 文件为例，向大家进行展示。

```
import json
with open('food.json', 'r') as f:
    data = json.load(f)
```

预览结果如图 3-10 所示。

```
In [34]: data[0]

Out[34]: {'description': 'Cheese, caraway',
          'group': 'Dairy and Egg Products',
          'id': 1008,
          'manufacturer': '',
          'nutrients': [{'description': 'Protein',
            'group': 'Composition',
            'units': 'g',
            'value': 25.18},
           {'description': 'Total lipid (fat)',
            'group': 'Composition',
            'units': 'g',
            'value': 29.2},
           {'description': 'Carbohydrate, by differenc
            'group': 'Composition',
            'units': 'g',
            'value': 3.06},
           {'description': 'Ash', 'group': 'Other', 'u
           {'description': 'Energy',
            'group': 'Energy',
```

图 3-10 food.json 数据预览展示图

以上就是文件 food.json 里面的内容。通常，理想的格式是像 pandas 那样标准的格式，该怎么处理呢？

仔细观察上面的 JSON 文件，可以发现有 5 个主键：description、group、id、manufacturer 和 nutrients。其中前 4 个键的取值是字符串，但 nutrients 的取值是一个列表，列表中包含的是另一个 JSON 格式的数据。

基于上述规律，先处理单独的 4 个主键（description、group、id 和 manufacturer），再处理主键 nutrients 的内容。

```
keys = ['id','description','group']
keys_df = pd.DataFrame(data, columns = keys)
nutrients = []
for i in data:
    temp = pd.DataFrame(i['nutrients'])
    temp['id'] = i['id']
    nutrients.append(temp)
nutrients = pd.concat(nutrients, ignore_index = True)
```

最终结果如图 3-11 所示。

In [36]: nutrients.head()

Out[36]:

	description	group	units	value	id
0	Protein	Composition	g	25.18	1008
1	Total lipid (fat)	Composition	g	29.20	1008
2	Carbohydrate, by difference	Composition	g	3.06	1008
3	Ash	Other	g	3.28	1008
4	Energy	Energy	kcal	376.00	1008

图 3-11　解析 food.json 中的 nutrients 数据

下面处理略微复杂一些的 JSON。

```
with open('report.json', 'r') as f:
    data = json.load(f)
```

【注意】

这个 JSON 文件涉及敏感信息，所以本书仅作代码示例，不提供文件示例。

预览结果如图 3-12 所示。

```
In [40]: data[0]

Out[40]: {'create_time': 1504088427000,
          'id': 1233,
          'modify_time': None,
          'mongo_id': None,
          'report_data': '{"trip_analysis":[{"called_cnt":"620","talk_s
         led_seconds":"58635","receive_cnt":"0","date_distribution":["2
         -03"],"detail":[{"called_cnt":"151","talk_seconds":"29980","ta
         1","receive_cnt":"0","month":"2017-08","call_cnt":"204","unknc
         {"called_cnt":90,"talk_seconds":14568,"talk_cnt":186,"msg_cnt'
         7-07","call cnt":96,"unknown cnt":0,"send cnt":0,"call seconds
```

图 3-12　report.json 数据预览展示图

下面展示解析 report_data 里的 trip_analysis 字段的代码示例。

```
trip_analysis = []
for i in data:
    str1 = i['report_data']
    dict1 = json.loads(str1)
    temp = pd.DataFrame(dict1['trip_analysis'])
    temp['user_id'] = i['user_id']
    trip_analysis.append(temp)
trip_analysis = pd.concat(trip_analysis, ignore_index = True)
```

最终结果如图 3-13 所示。

```
In [43]: trip_analysis.head(1)
Out[43]:
```

	call_cnt	call_seconds	called_cnt	called_seconds	date_distribution
0	847	66569	620	58635	[2017-08, 2017-07, 2017-06, 2017-05, 2017-04, ...

图 3-13　解析 report.json 中的 trip_analysis 数据

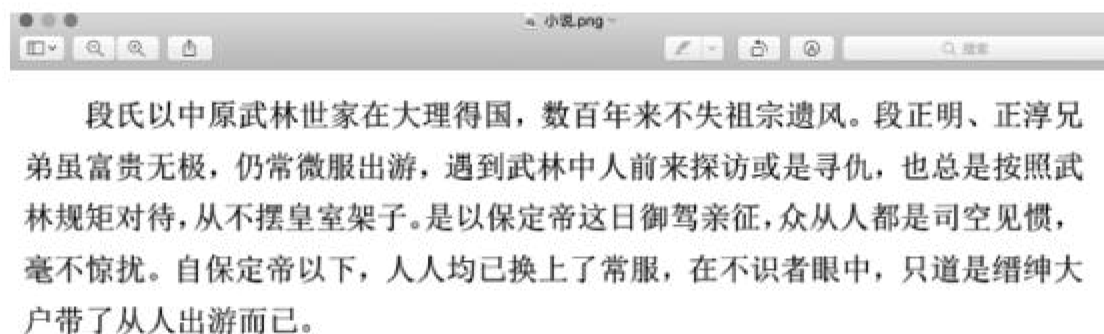
3.6　OCR 文字识别



视频讲解

先给大家看一张带有文字内容的图片，如图 3-14 所示。

将图片识别成文字，如图 3-15 所示。



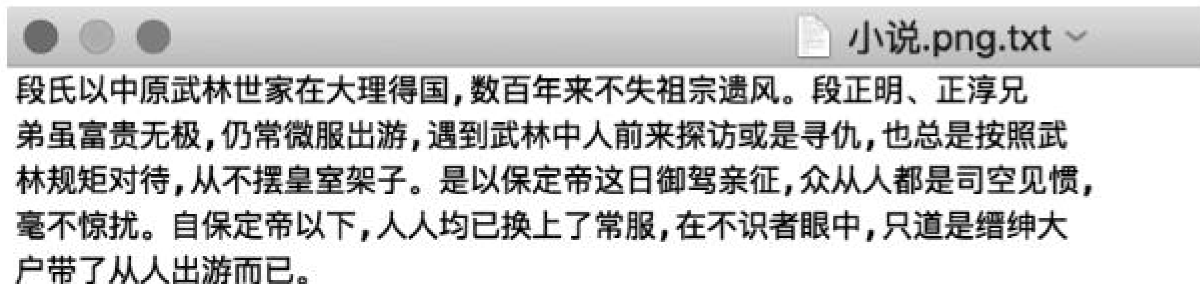
段氏以中原武林世家在大理得国，数百年来不失祖宗遗风。段正明、正淳兄弟虽富贵无极，仍常微服出游，遇到武林中人前来探访或是寻仇，也总是按照武林规矩对待，从不摆皇室架子。是以保定帝这日御驾亲征，众从人都是司空见惯，毫不惊扰。自保定帝以下，人人均已换上了常服，在不识者眼中，只道是缙绅大户带了从人出游而已。

图 3-14　小说.png 图片展示图

```
zhaikundeiMac:~ zhaikun$ python ocr.py
请输入文件名: 小说.png
已收到, 正在处理, 请稍后...
处理完成
zhaikundeiMac:~ zhaikun$
```

图 3-15　图片识别程序运行展示图

下面是转化后的效果，如图 3-16 所示。


小说.png.txt

段氏以中原武林世家在大理得国，数百年来不失祖宗遗风。段正明、正淳兄弟虽富贵无极，仍常微服出游，遇到武林中人前来探访或是寻仇，也总是按照武林规矩对待，从不摆皇室架子。是以保定帝这日御驾亲征，众从人都是司空见惯，毫不惊扰。自保定帝以下，人人均已换上了常服，在不识者眼中，只道是缙绅大户带了从人出游而已。

图 3-16　图片识别为文字之后的 txt 文档展示图

是不是很酷！这是怎么做的呢？其实操作很简单，写一段简单的代码，调用百度 API 即可。

先打开网址：https://login.bce.baidu.com/，登录百度账号，如图 3-17 所示。



图 3-17　登录百度云页面展示图

然后单击“创建应用”按钮，就能获取 AppID、API Key、Secret Key，如图 3-18 和图 3-19 所示。



图 3-18　创建文字识别应用

应用名称	AppID	API Key	Secret Key

图 3-19　应用名称及对应的 AppID、API Key、Secret Key 展示图

然后安装 baidu-aip，安装语句为：pip install baidu-aip，安装教程详见附录 F。安装好后，代码如下。

```
from aip import AipOcr
import codecs
import os
os.chdir('../data')
def ocr(path):
    with open(path, 'rb') as f:
        return f.read()
def main():
    file_name = str(input('请输入文件名:'))
    print ('已收到,正在处理,请稍后...')
    app_id = '你自己的 id'
    api_key = '你自己的 api_key '
    secret_key = '你自己的 secret_key '
    client = AipOcr(app_id, api_key, secret_key)
    image = ocr(file_name)
    dict1 = client.general(image)
    with codecs.open(file_name + ".txt", "w", "utf-8") as f:
        for i in dict1['words_result']:
            f. write(str(i['words']) + "\r\n")
    print ('处理完成')
if __name__ == '__main__':
    main()
```

通过运行以上代码就可以成功调用百度的 API,从而进行 OCR 识别了。除了普通的文字识别,还能进行以下特殊文件识别,如图 3-20 所示。

- 网络图片文字识别
- 身份证识别
- 银行卡识别
- 驾驶证识别
- 行驶证识别
- 车牌识别
- 营业执照识别
- 表格文字识别
- 通用票据识别

图 3-20　百度 OCR 的其他文件识别展示图

具体介绍的链接地址为:https://cloud.baidu.com/doc/OCR/OCR-API.html。

3.7 pyecharts



视频讲解

ECharts是一个使用JavaScript实现的开源可视化库，可以流畅地运行在PC端和移动设备上，提供直观的、交互丰富的、可高度个性化定制的数据可视化图表，官网链接地址为：http://echarts.baidu.com/。用ECharts生成的图的可视化效果非常棒，为了与Python进行对接，同时方便在Python中直接使用数据生成图，国内开发人员开发出了第三方包：pyecharts。

首先安装pyecharts，安装语句为：pip install pyecharts，安装教程详见附录F。安装好后，打开Jupyter Notebook。pyecharts有交互效果，所以需要用Jupyter Notebook运行以下代码。

先读取数据：

```
import pandas as pd
import os
os.chdir('../data')
data = pd.read_excel('房价与工资.xlsx')
```

数据预览如图3-21所示。

In [3]: data.head(3)

Out[3]:

	城市	省份	房屋均价	人均工资
0	北京市	北京	63239	9240
1	重庆市	重庆	10351	5962
2	广州市	广东	30894	7409

图3-21 房价与工资.xlsx数据预览展示图

下面指定x轴和y轴：

```
x = list(data['城市'])
y1 = list(data['房屋均价'])
y2 = list(data['人均工资'])
```

先看如何绘制垂直条形图，代码如下，结果如图3-22所示。

```
from pyecharts import Bar
bar = Bar()
```

```
bar.add("房屋均价(元)", x, y1, mark_point = ["max", "min"], mark_line = ["average"], is_
label_show = True)
bar.add("人均工资(元)", x, y2)
bar
```

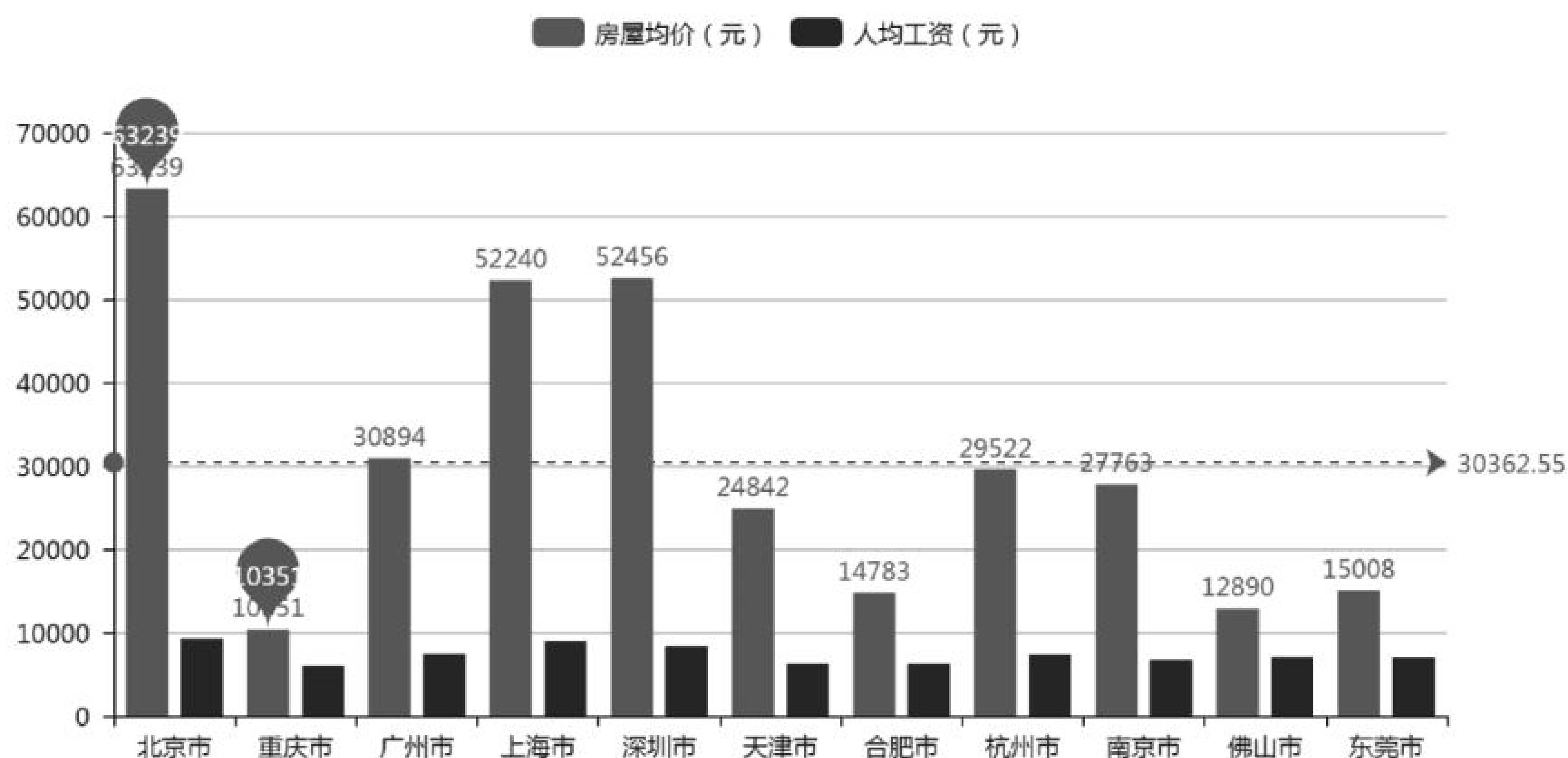


图 3-22　房价与工资.xlsx 数据垂直条形图

图例是可以交互的,单击图 3-22 正上方的“房屋均价(元)”,可得如图 3-23 所示的交互图。



图 3-23　房价与工资.xlsx 数据交互图

再看如何绘制水平条形图,代码如下,结果如图 3-24 所示。

```
from pyecharts import Bar
bar = Bar()
bar.add("房屋均价(元)", x, y1)
bar.add("人均工资(元)", x, y2, is_convert = True)
bar
```

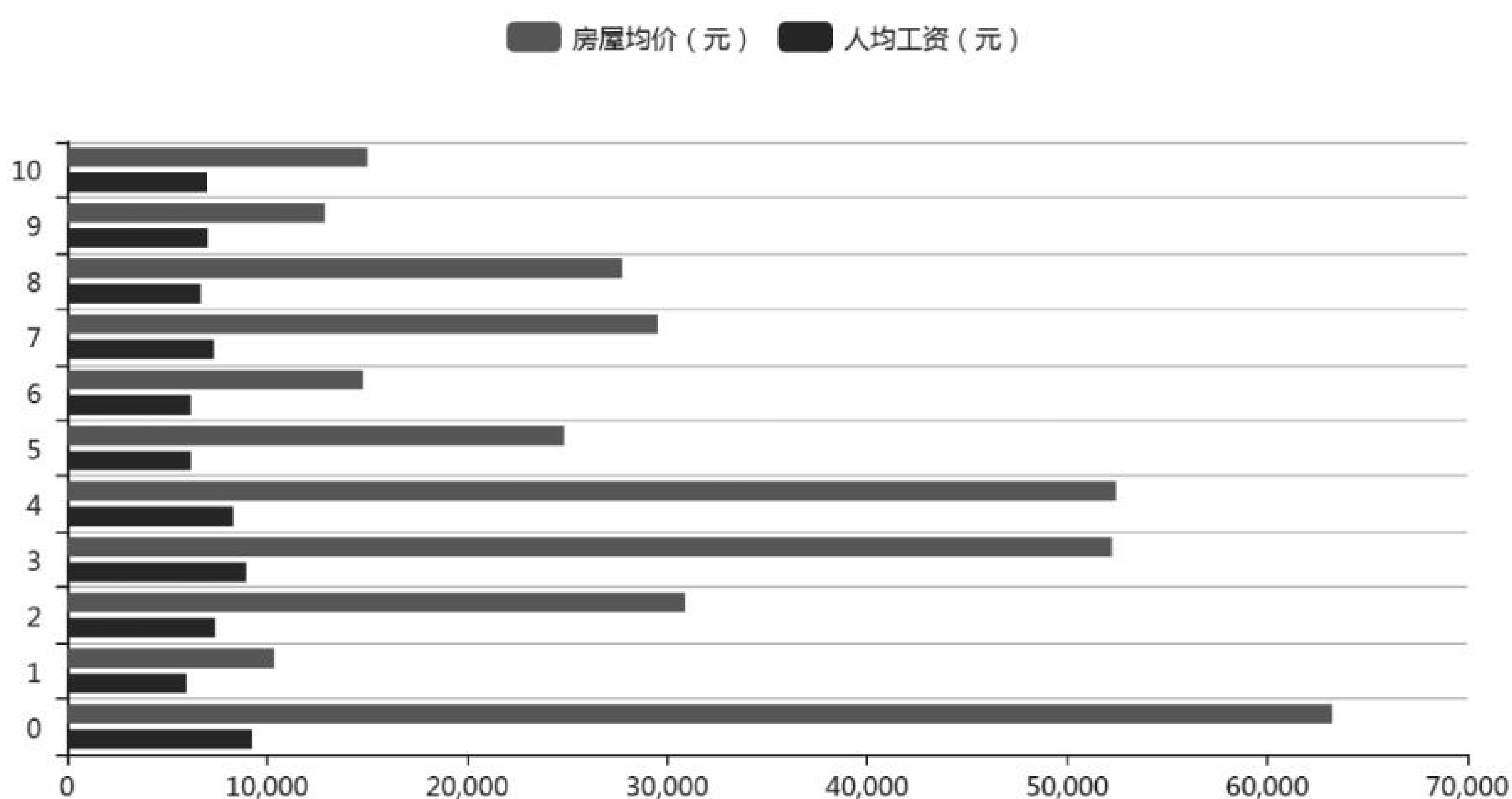


图 3-24 房价与工资.xlsx 数据水平条形图

再看如何绘制直方图,代码如下,结果如图 3-25 所示。

```
from pyecharts import Bar
bar = Bar()
bar.add('房屋均价(元)', x, y1, bar_category_gap = 10, is_label_show = True)
bar
```

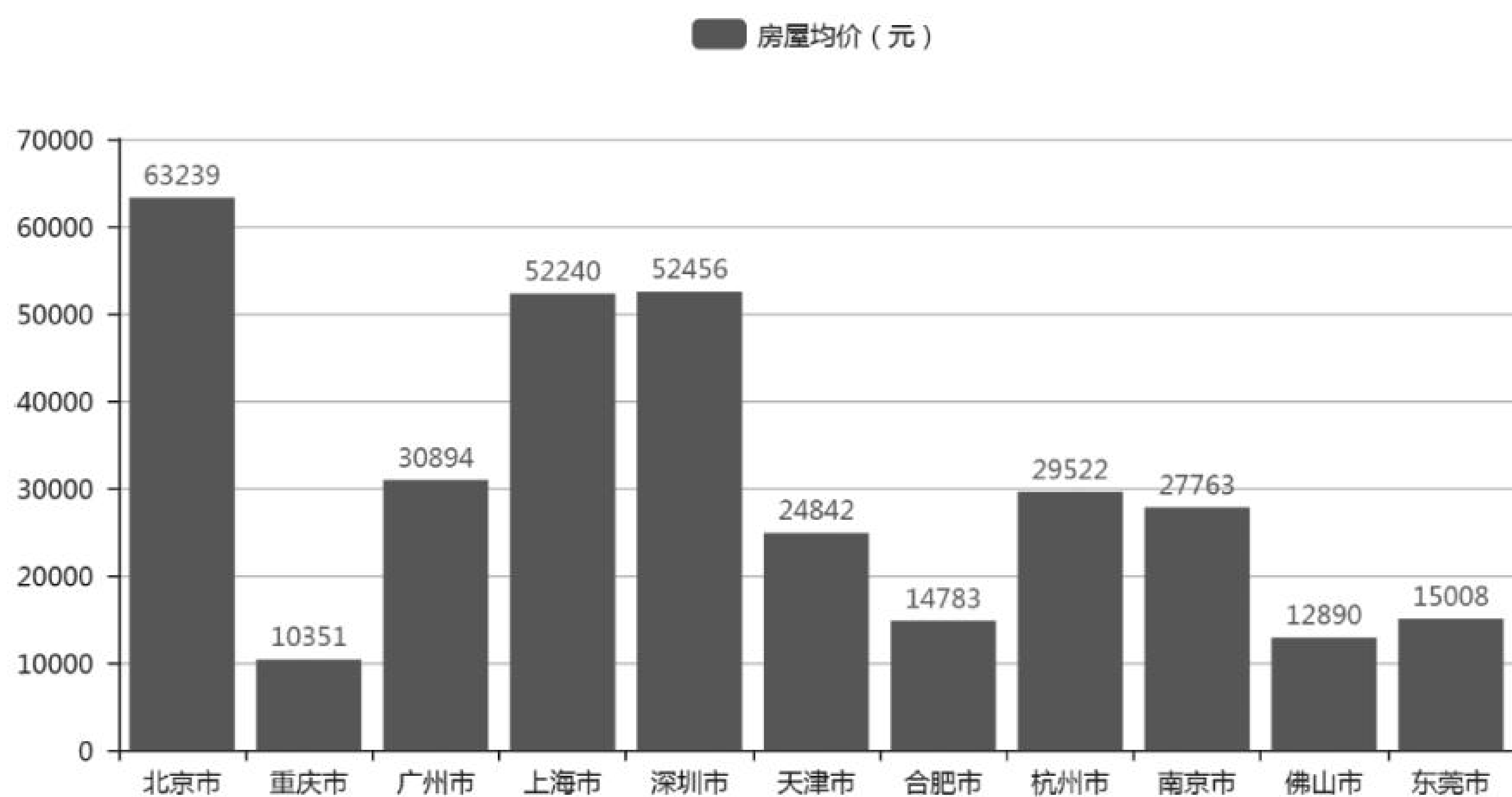


图 3-25 房价与工资.xlsx 的房屋均价直方图

再看如何绘制折线图,代码如下,结果如图 3-26 所示。

```
from pyecharts import Line
line = Line()
```

```
line.add('房屋均价(元)', x, y1, is_label_show = True)
line.add('人均收入(元)', x, y2)
line
```

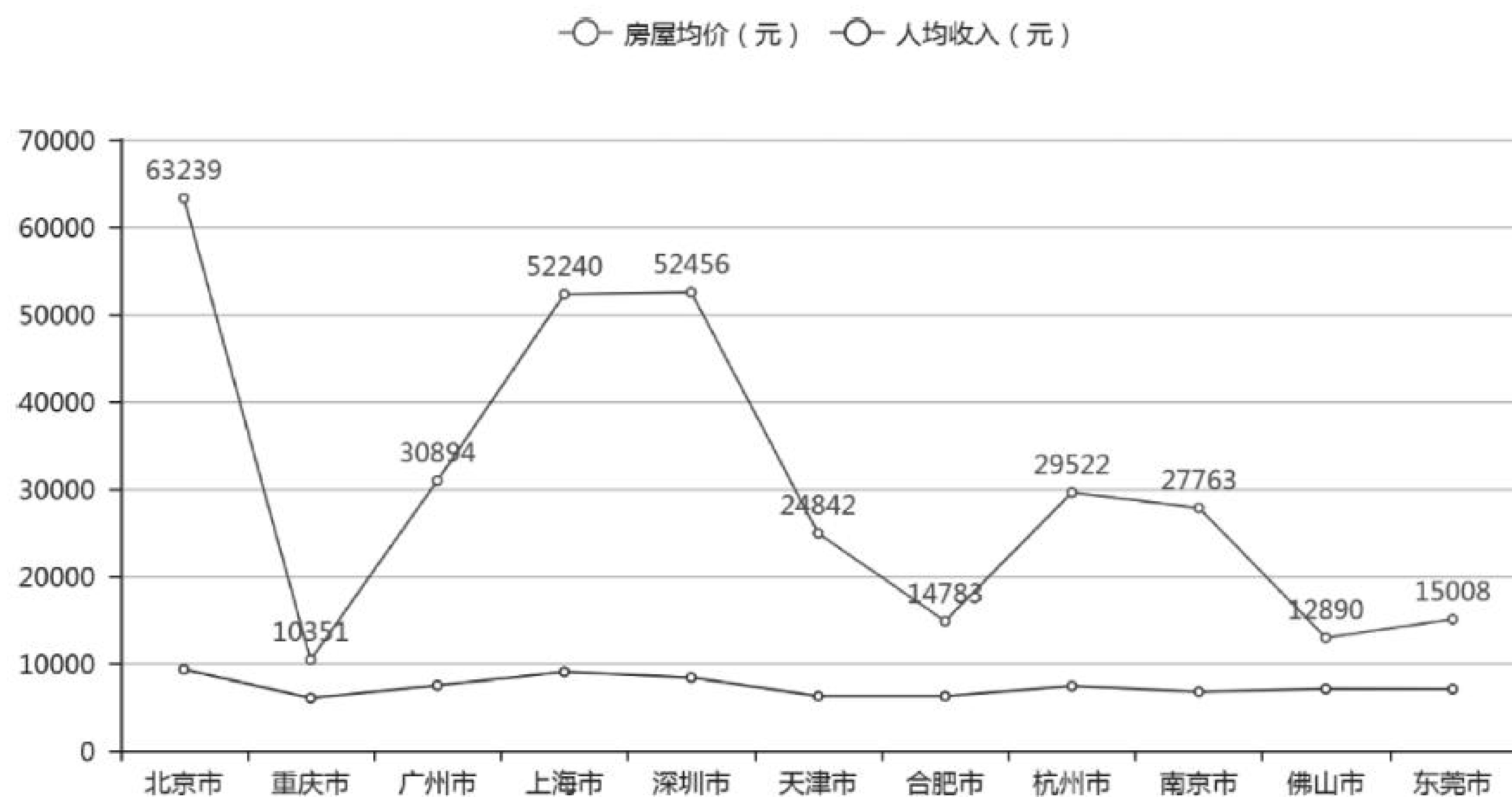


图 3-26　房价与工资.xlsx 的数据折线图

再看如何绘制饼图,代码如下,结果如图 3-27 所示。

```
from pyecharts import Pie
pie = Pie()
pie.add('房屋均价(元)', x, y1, is_label_show = True, legend_orient = 'vertical', legend_pos
 = 'left')
pie
```

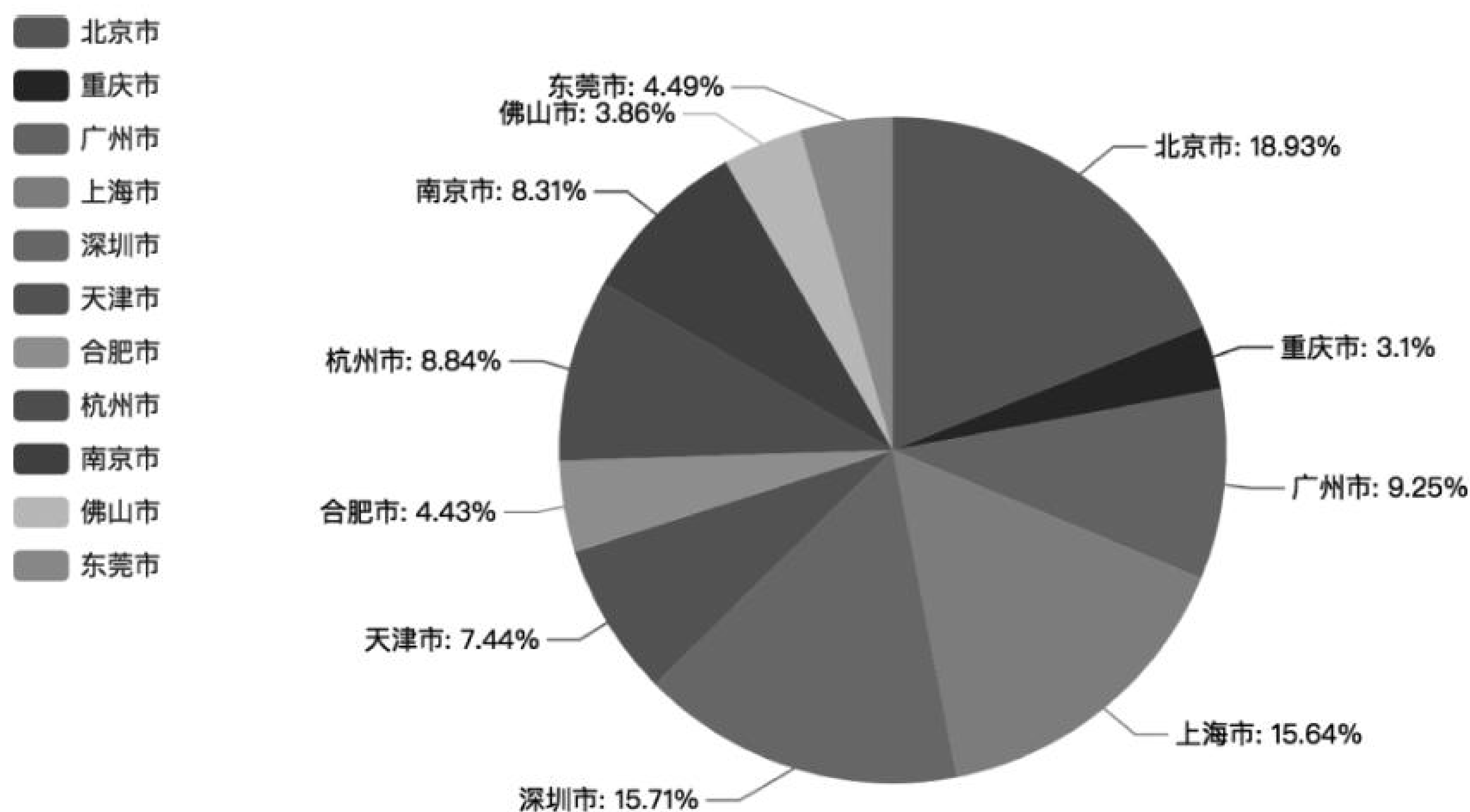


图 3-27　房价与工资.xlsx 的房屋饼图

也可将条形图和折线图绘制在同一个图形里,代码如下,结果如图 3-28 所示。

```
from pyecharts import Bar, Line, Grid
bar = Bar()
bar.add("房屋均价(元)", x, y1, is_label_show = True)
bar.add("人均工资(元)", x, y2)
line = Line()
line.add('房屋均价(元)', x, y1, is_label_show = True)
line.add('人均工资(元)', x, y2)
grid = Grid()
grid.add(bar, grid_bottom = "60 % ")
grid.add(line, grid_top = "60 % ")
grid
```

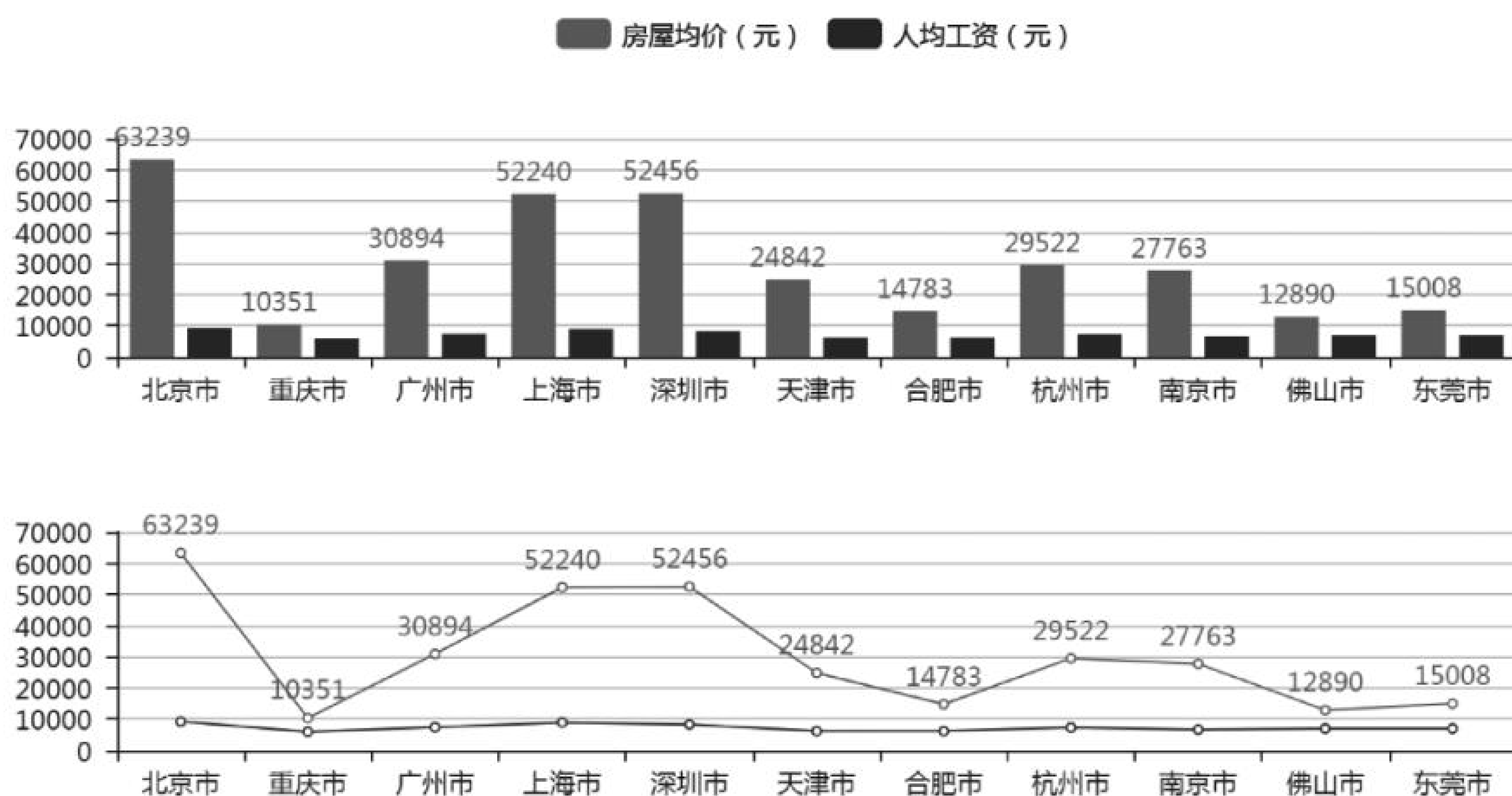


图 3-28 房价与工资.xlsx 的数据条形图和折线图

在条形图上叠加折线图,代码如下,结果如图 3-29 所示。

```
from pyecharts import Overlap, Bar, Line, Grid
bar = Bar()
bar.add('房屋均价(元)', x, y1)
bar.add('人均工资(元)', x, y2)
line = Line()
line.add('房屋均价(元)', x, y1, is_label_show = True)
line.add('人均工资(元)', x, y2, is_label_show = True)
overlap = Overlap()
overlap.add(bar)
overlap.add(line)
overlap
```

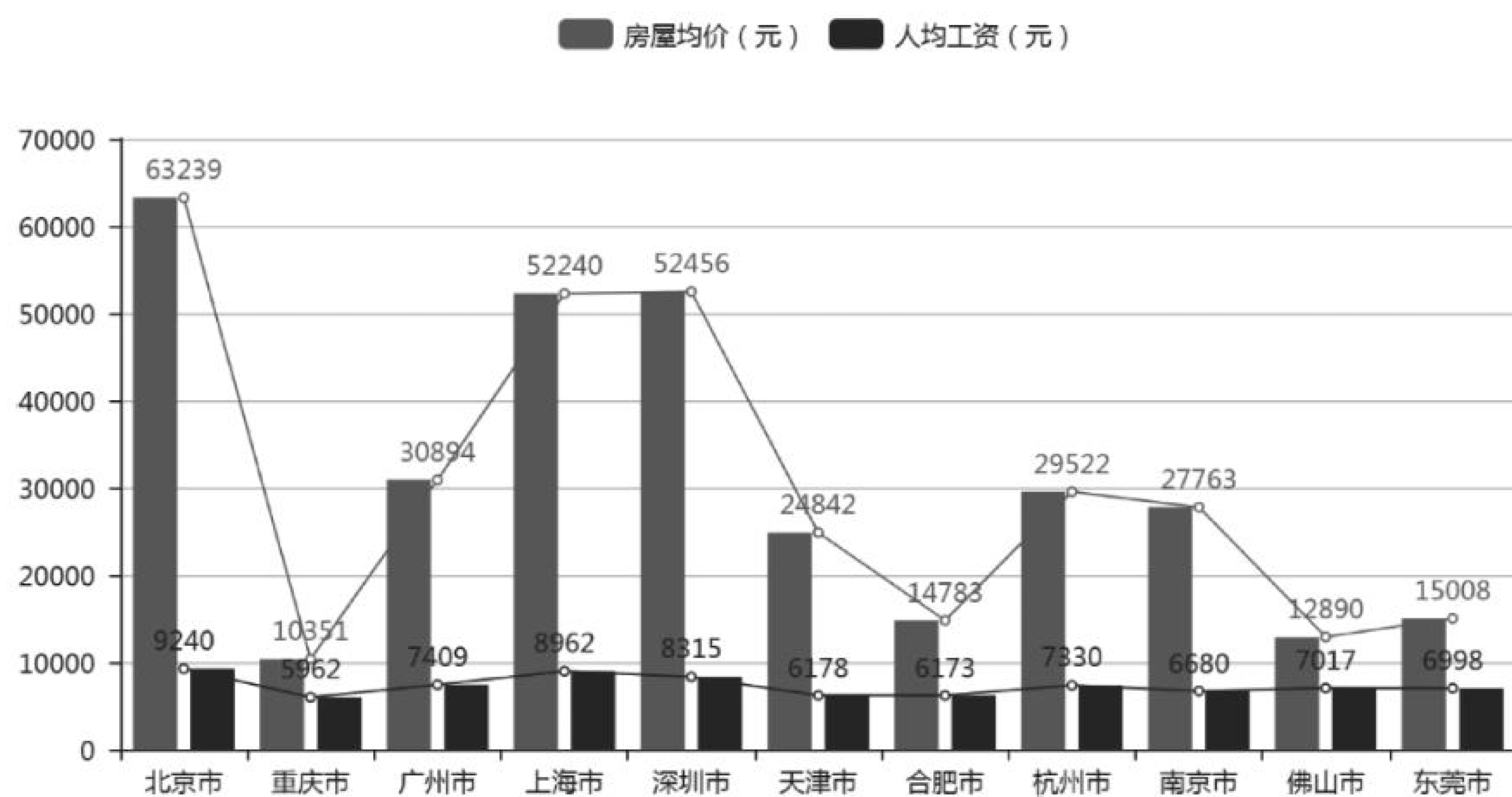


图 3-29　房价与工资.xlsx 的数据条形图叠加折线图

使用 pyecharts 也可创建仪表盘图，代码如下，结果如图 3-30 所示。

```
from pyecharts import Gauge
gauge = Gauge()
gauge.add('', '', '80', scale_range=[0, 100], angle_range=[180, 0])
gauge
```

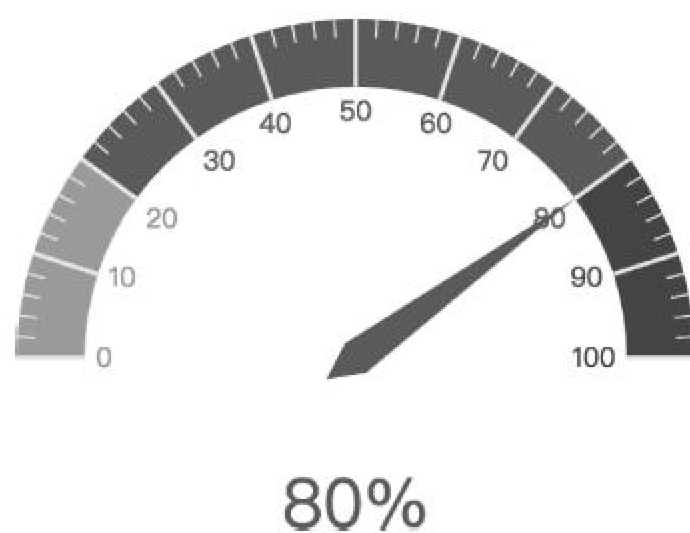


图 3-30　pyecharts 创建的仪表盘图

使用 pyecharts 也可创建城市散点图，感兴趣的读者可以尝试运行，看看运行结果。

```
import numpy as np
temp = np.array(data[['城市','房屋均价']]).tolist()
a = []
for i in temp:
    z = tuple(i)
    a.append(z)
from pyecharts import Geo
```

```
geo = Geo("全国主要城市房价散点图", title_color = "#fff", title_pos = "center", width =
800, height = 600, background_color = '#404a59')
city, value = geo.cast(a)
geo.add("", city, value, visual_range = [10000, 50000], visual_text_color = "#fff",
symbol_size = 15, is_visualmap = True)
geo
```

使用 pyecharts 也可创建城市散点涟漪图，代码如下：

```
from pyecharts import Geo
geo = Geo("全国主要城市房价散点涟漪图", title_color = "#fff", title_pos = "center",
width = 800, height = 600, background_color = '#404a59')
city, value = geo.cast(a)
geo.add("", city, value, type = "effectScatter", is_random = True, effect_scale = 5)
geo
```

使用 pyecharts 也可创建热力图，代码如下：

```
from pyecharts import Geo
geo = Geo("全国主要城市房价热力图", title_color = "#fff", title_pos = "center", width =
800, height = 600, background_color = '#404a59')
city, value = geo.cast(a)
geo.add("", city, value, type = "heatmap", is_visualmap = True, visual_range = [10000,
20000], visual_text_color = '#fff')
geo
```

使用 pyecharts 也可创建全国省份地图，代码如下：

```
from pyecharts import Map
city = list(data['省份'])
value = list(data['房屋均价'])
map = Map( width = 800, height = 600)
map.add("", city, value, maptype = 'china', is_label_show = True, is_visualmap = True,
visual_range = [5000, 60000], visual_text_color = '#000')
map
```

使用 pyecharts 也可创建指定省份城市图，代码如下：

```
from pyecharts import Map
city = np.array(data['城市']).tolist()
value = np.array(data['房屋均价']).tolist()
map = Map( width = 800, height = 600)
map.add("",city,value,maptype = '广东', is_label_show = True, is_visualmap = True, visual_
range = [10000, 20000],visual_text_color = '#000')
```

```
map
```

从上面的各种图形中可以看出,使用 pyecharts 作的图确实很漂亮,以后做数据分析展示图的时候,可以尝试使用 pyecharts 进行作图,然后将这些图片放到 PPT 报告里,从而提高报告的展现力。

3.8 stats 简单统计分析

Python 虽然不像 R 语言那样,专为统计而生,但是它也可以进行统计分析与检验。scipy 库中的 stats 包就能实现大多数的统计功能。

学生的考试成绩会受到多种因素的影响,如:是否分场、试卷难易程度和临场发挥情况。本次仅考虑“是否分场”这一因素。下面以北京市日坛实验中学的初三生物考试成绩为例,分析在考场合并前后,学生的成绩有无显著差异。

```
import pandas as pd
import matplotlib.pyplot as plt
from pylab import mpl
mpl.rcParams['font.sans-serif'] = ['SimHei']  #指定默认字体
mpl.rcParams['axes.unicode_minus'] = False     #解决保存图像是负号'-'显示为方块的问题
import os
os.chdir('/Users/zhaikun/Desktop')
data = pd.read_excel('成绩.xls')
```

下面预览一下数据,如图 3-31 所示。

```
In [4]: data.head()
Out[4]:
```

	分场1	分场2	分场3	分场4	分场_月考	合场1	合场2	合场3	合场_期中
0	27	26	29	38	30	24	29	37	32
1	30	38	38	40	45	38	38	36	44
2	43	49	47	46	43	48	48	42	50
3	30	34	29	41	32	36	29	38	15
4	39	45	40	44	48	50	40	40	48

图 3-31 初三生物考试成绩数据预览图

首先计算出每个分场的考试成绩平均值,再计算合并考场之后的成绩平均值,代码如下所示。

```
data['分场平均值'] = data.ix[:,0:5].mean(1)
data['合场平均值'] = data.ix[:,5:9].mean(1)
temp = data[['分场平均值', '合场平均值']]
temp.plot()
plt.show()
```

计算考试成绩平均值之后,可以通过可视化,查看每个考生的成绩差异,结果如图 3-32 所示。

```
In [10]: temp.plot()
         plt.show()
```

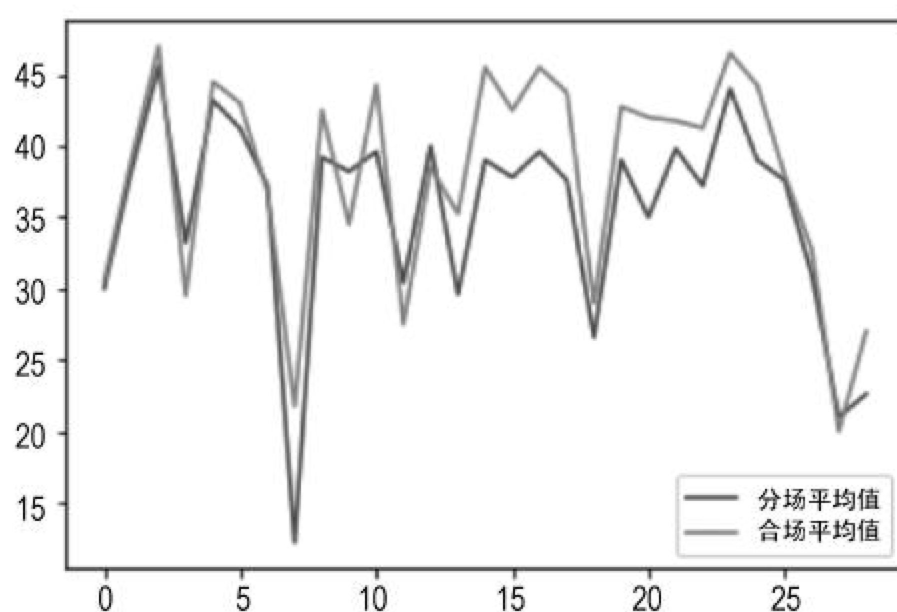


图 3-32　生物考试成绩差异图

由图 3-32 可以发现,合并考场之后,考试平均成绩明显比分场考试平均成绩高。

下面进行“相关样本的 t 检验”,原假设是样本均值相等,即考生在考场合并前后的考试成绩应该是相等的。代码如下所示,得到的结果如图 3-33 所示。

```
from scipy import stats
stats.ttest_rel(temp['分场平均值'], temp['合场平均值'])
```

```
In [11]: from scipy import stats
         stats.ttest_rel(temp['分场平均值'], temp['合场平均值'])
Out[11]: Ttest_relResult(statistic=-4.0781602275865065, pvalue=0.0003406411787764341)
```

图 3-33　t 检验结果图

从计算结果中,可以发现 $P<0.05$,即原假设不成立,就是考场合并前后,考生的考试成绩均值差异显著。

stats 包除了可以进行 t 检验以外,还可以进行卡方检验、方差分析等统计上常用的检验手段,这里不进行详细介绍,感兴趣的读者可自行查阅相关资料。

3.9 小结

本章主要介绍了与 Python 语言相关的实际应用，这些应用均使用了第三方包，主要包括数据的连接、词云图、社交网络、JSON 解析、OCR 文字识别以及 pyecharts 绘图等。通过这些实际应用，展示了 Python 作为“胶水语言”的魅力，感兴趣的读者可以查找相关资料，进行进一步的学习与探索。

第 4 章



异常样本识别

什么是不平衡数据？比如，10 万个普通人中找出 10 个恐怖分子，10 万个正常还款的客户中找出 10 个欺诈客户，这些都属于不平衡数据。由于极少数异常样本可能带来很大的损失，在工作中，主要任务就是要在这些不平衡数据中找到那些占比极少的异常样本。

4.1 逻辑回归、交叉验证与欠采样



视频讲解

首先，导入包和加载数据：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data')
data = pd.read_csv('data.csv')
```

在 Jupyter Notebook 中运行代码，其数据预览结果如图 4-1 所示。

```
In [2]: data.head()
Out[2]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206

5 rows × 31 columns

```
In [3]: data.shape
Out[3]: (284807, 31)
```

图 4-1 data.csv 数据预览展示图

这个数据集是 284807 行，31 列的数据。

```
data['Class'].value_counts()
0    284315
1       492
Name: Class, dtype: int64
```

目标变量'Class'非常不平衡，异常客户数据只有 492 个，异常占比为 492÷284807=1.73%。

先对变量列'Amount'，'Time'进行标准化处理。

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data[['Amount', 'Time']] = sc.fit_transform(data[['Amount', 'Time']])
```

如果对数据不进行任何处理，而直接用逻辑回归建模，来看看模型效果。

```
allFeatures = list(data.columns)
allFeatures.remove('Class')
X = data[allFeatures]
y = data['Class']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
from sklearn.linear_model import LogisticRegression as LR
lr = LR()
lr.fit(X_train, y_train)
from sklearn import metrics
y_test_label = lr.predict(X_test)
y_test_value = lr.predict_proba(X_test)[:, 1]
print("测试集准确率是: {:.2%}".format(metrics.accuracy_score(y_test, y_test_label)))
print("测试集 AUC 是: {:.4}".format(metrics.roc_auc_score(y_test, y_test_value)))
from sklearn.metrics import classification_report
print(classification_report(y_test, y_test_label))
```

运行结果如下：

```
测试集准确率是: 99.92 %
测试集 AUC 是: 0.9657
             precision    recall  f1 - score   support
          0       1.00      1.00       1.00     85308
          1       0.84      0.58       0.68       135
avg / total       1.00      1.00       1.00     85443
```

看起来效果还不错，但是能进行优化吗？

这里，首先要明白目标是什么？衡量分类器的主要指标是什么？在这个不平衡的异常样本检测案例中，衡量的指标主要是 recall，以此来衡量，模型表现显然不是很好。

那么关于不平衡样本，可以试试欠采样的方法。这里要使用到第三方包 imblearn，安装语句为：pip install imblearn。

```
allFeatures = list(data.columns)
allFeatures.remove('Class')
X = data[allFeatures]
y = data['Class']
n_pos_sample = y[y == 0].shape[0]
n_neg_sample = y[y == 1].shape[0]
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(ratio = {0:(4 * n_neg_sample),1:(n_neg_sample)},random_state = 1)
X, y = rus.fit_sample(X, y)
data_X = pd.DataFrame(X, columns = [allFeatures])
data_y = pd.DataFrame(y, columns = ['target'])
data = pd.concat([data_X,data_y], axis = 1)
```

在这个重新构成的新样本中，目标变量 target 的水平分布是：

```
data['target'].value_counts()
0    1968
1     492
Name: target, dtype: int64
```

下面重新建模：

```
X = data[allFeatures]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size = 0.3, random_state = 1)
from sklearn.linear_model import LogisticRegression as LR
```

```
lr = LR()
lr.fit(X_train, y_train)
from sklearn import metrics
y_test_label = lr.predict(X_test)
y_test_value = lr.predict_proba(X_test)[:, 1]
print("测试集准确率是::.2%}".format(metrics.accuracy_score(y_test, y_test_label)))
print("测试集 AUC 是:{:.4}".format(metrics.roc_auc_score(y_test, y_test_value)))
from sklearn.metrics import classification_report
print(classification_report(y_test, y_test_label))
```

运行结果如下：

```
测试集准确率是:97.15%
测试集 AUC 是:0.9805
              precision    recall  f1-score   support
          0        0.97      1.00      0.98       588
          1        0.98      0.87      0.93       150
avg / total        0.97      0.97      0.97       738
```

召回率 recall 是 0.87,模型表现有了很大的提升。

下面再试试 5 折交叉验证：

```
from sklearn.cross_validation import KFold
from sklearn.metrics import recall_score
fold = KFold(len(y_train), 5, shuffle=False)
c_param_range = [0.01, 0.1, 1, 10, 100]
results_table = pd.DataFrame(columns = ['C值', '平均召回率得分'])
results_table['C值'] = c_param_range
j = 0
for c_param in c_param_range:
    print('C值: ', c_param)
    recall_accs = []
    for iteration, indices in enumerate(fold, start=1):
        lr = LR(C = c_param, penalty = 'l1')
        X_train = X_train.reset_index(drop=True)
        y_train = y_train.reset_index(drop=True)
        lr.fit(X_train.iloc[indices[0],:], y_train.iloc[indices[0]].values.ravel())
        y_pred = lr.predict(X_train.iloc[indices[1],:].values)
        recall_acc = recall_score(y_train.iloc[indices[1]].values, y_pred)
        recall_accs.append(recall_acc)
        print('迭代次数', iteration,':召回率得分= ', recall_acc)
    results_table.ix[j, '平均召回率得分'] = np.mean(recall_accs)
    j += 1
    print('平均召回率得分: ', np.mean(recall_accs))
best_c = results_table.loc[results_table['平均召回率得分'].idxmax()]['C值']
```

```
print('交叉验证最好的C值是', best_c)
```

运行结果如下：

```
C值:0.01
迭代次数 1:召回率得分 = 0.7972972972972973
迭代次数 2:召回率得分 = 0.8769230769230769
迭代次数 3:召回率得分 = 0.875
迭代次数 4:召回率得分 = 0.7878787878787878
迭代次数 5:召回率得分 = 0.8461538461538461
平均召回率得分:0.8366506016506016
C值:0.1
迭代次数 1:召回率得分 = 0.8243243243243243
迭代次数 2:召回率得分 = 0.9076923076923077
迭代次数:3:召回率得分 = 0.875
迭代次数:4:召回率得分 = 0.8484848484848485
迭代次数 5:召回率得分 = 0.8615384615384616
平均召回率得分:0.8634079884079885
C值:1
迭代次数 1:召回率得分 = 0.8243243243243243
迭代次数 2:召回率得分 = 0.9230769230769231
迭代次数 3:召回率得分 = 0.875
迭代次数 4:召回率得分 = 0.8484848484848485
迭代次数 5:召回率得分 = 0.8923076923076924
平均召回率得分:0.8726387576387576
C值:10
迭代次数 1:召回率得分 = 0.8378378378378378
迭代次数 2:召回率得分 = 0.9230769230769231
迭代次数 3:召回率得分 = 0.8888888888888888
迭代次数 4:召回率得分 = 0.8787878787878788
迭代次数 5:召回率得分 = 0.8923076923076924
平均召回率得分:0.8841798441798442
C值:100
迭代次数 1:召回率得分 = 0.8378378378378378
迭代次数 2:召回率得分 = 0.9230769230769231
迭代次数 3:召回率得分 = 0.8888888888888888
迭代次数:4:召回率得分 = 0.8787878787878788
迭代次数 5:召回率得分 = 0.8923076923076924
平均召回率得分:0.8841798441798442
交叉验证最好的C值是 10.0.
```

下面把用交叉验证找到的C值用于模型中，先用欠采样后的数据训练模型，再把训练好的模型用于原始数据。

```
lr = LR(C = best_c, penalty = 'l1')
```

```
lr.fit(X_train, y_train)
data = pd.read_csv('data.csv')
allFeatures = list(data.columns)
allFeatures.remove('Class')
X = data[allFeatures]
y = data['Class']
y_label = lr.predict(X)
print(classification_report(y, y_label))
```

运行结果如下：

```
             precision    recall   f1 - score   support
          0       1.00      0.99         1.00    284315
          1       0.17      0.89         0.29       492
avg / total       1.00      0.99         1.00    284807
```

召回率 recall 有很大的提升，达到了 0.89。

```
y_value = lr.predict_proba(X)[:, 1]
print("AUC 是: {:.4}".format(metrics.roc_auc_score(y, y_value)))
```

运行结果如下：

```
AUC 是:0.9855
```

可以看出模型的表现很好。

在这个不平衡样本案例中，可以学到以下两点。

(1) 在不平衡的数据集中，准确率不再是衡量的指标。根据场景的不同，衡量指标可能是 racall，也可能是 precision，还可能是 f1-score，不要仅局限于 auc 和 ks。

(2) 对不平衡数据集，可以采用过采样或欠采样的方法，训练出在平衡样本(采样之后的样本)下的模型，然后把模型用于原始的不平衡数据中。

4.2 基于分布的异常样本识别


视频讲解

除了欠采样的异常样本识别，还可以根据变量取值的分布情况，进行异常样本识别，代码如下：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data')
dataset = pd.read_csv('data.csv')
'''连续单变量密度分布图'''
col = ['Amount', 'Time']
for i in col:
    plt.figure(figsize = (5,5))
    sns.distplot(dataset[i].dropna())
    plt.show()
```

运行结果如图 4-2 所示。

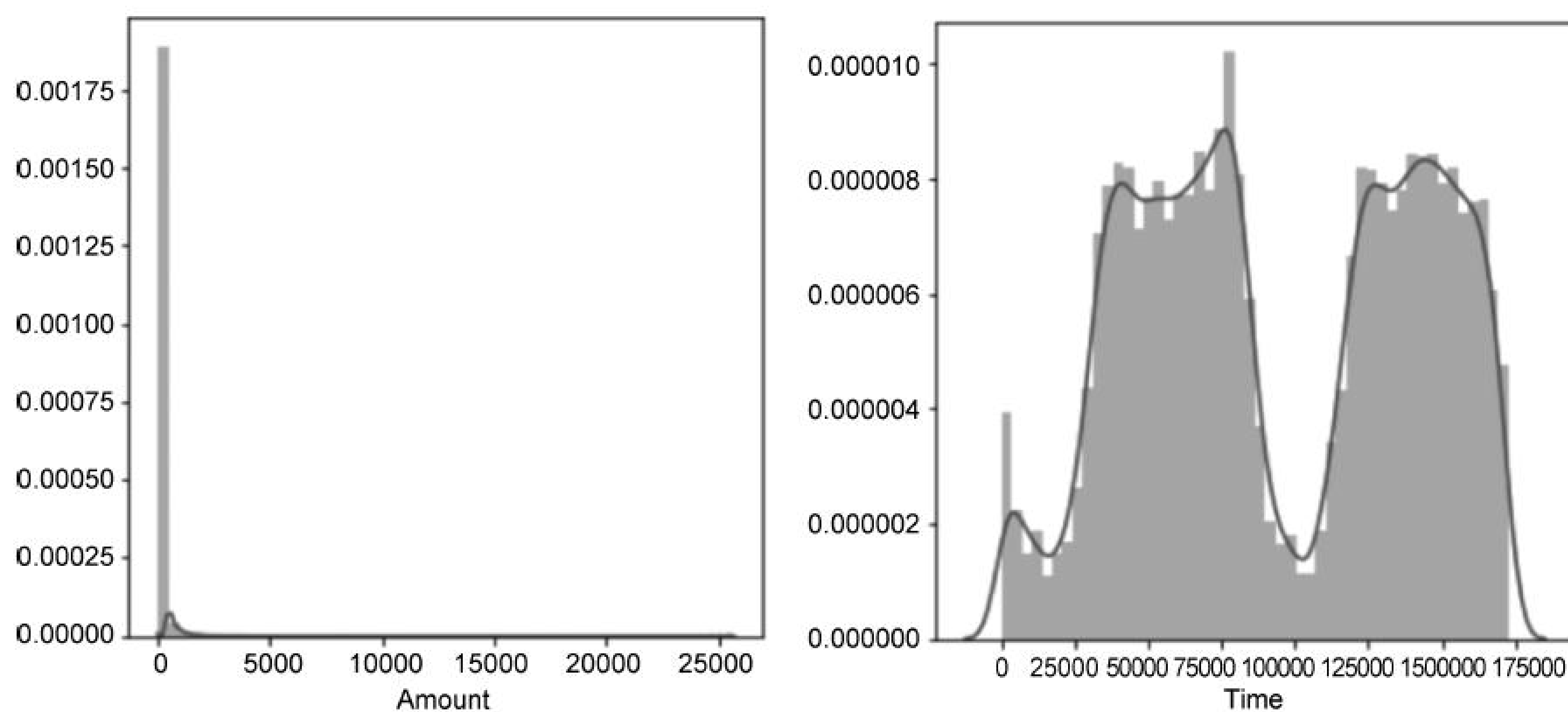


图 4-2　Amount 和 Time 变量的密度分布图

从图 4-2 中可以看出数据分布明显是偏斜的，那么作变量进行对数转换。

```
dataset['Amount'] = np.log(dataset['Amount'] + 1)
dataset['Time'] = np.log(dataset['Time'] + 1)
```

对数转换后的分布图，如图 4-3 所示。

先来回顾一下目标变量水平分布。

```
dataset['Class'].value_counts()
0    284315
1       492
Name: Class, dtype: int64
```

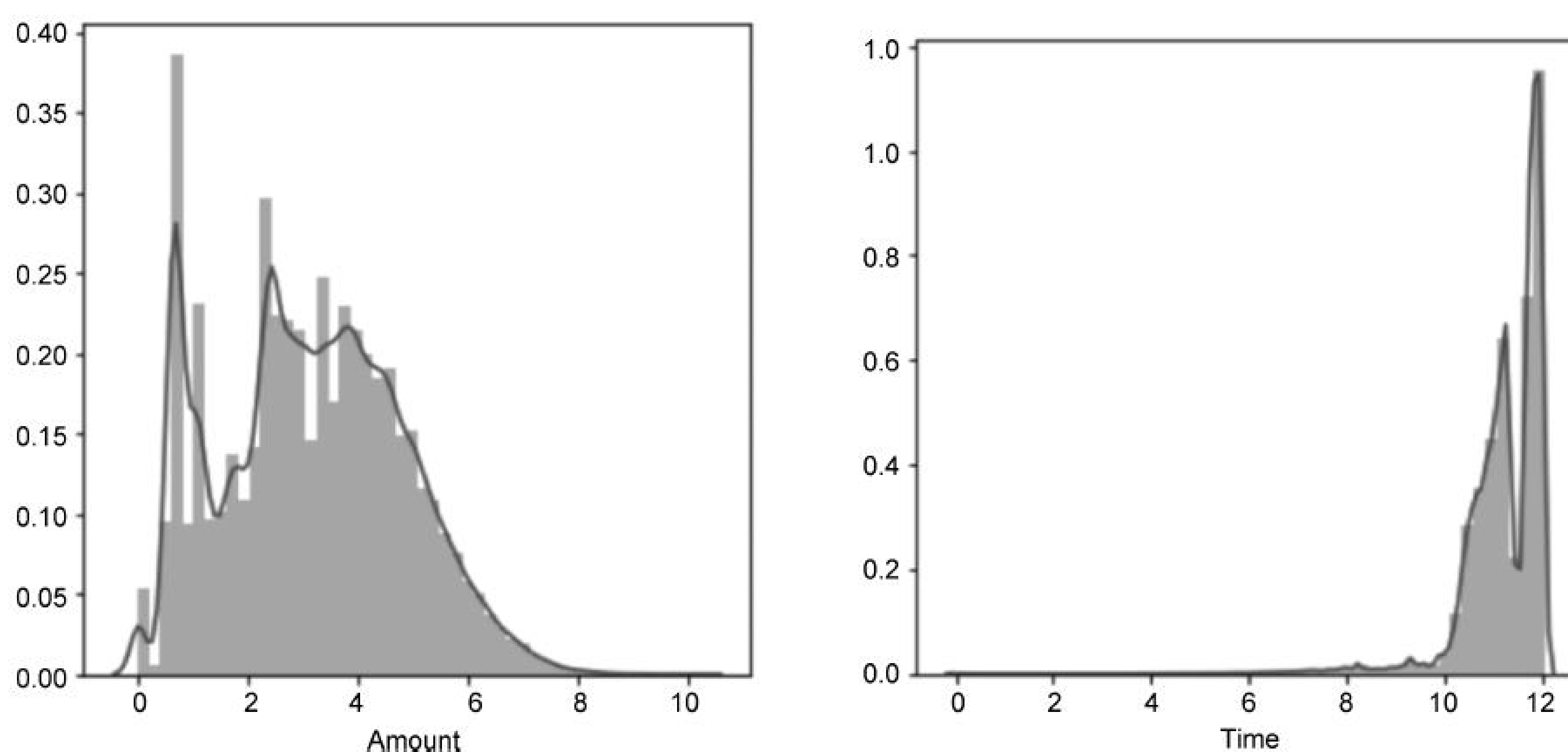


图 4-3 Amount 和 Time 变量的对数密度分布图

由目标变量水平分布可以看出，Class 等于 1 的样本数占比为 492÷284807≈0.1727%，是严重不平衡数据。

下面进行数据集切分：

```
normal = dataset[dataset['Class'] == 0]
anomaly = dataset[dataset['Class'] == 1]
from sklearn.model_selection import train_test_split as sp
train, test = sp(normal, test_size = 0.2)
normal_valid, normal_test = sp(test, test_size = 0.5)
anormal_valid, anormal_test = sp(anomaly, test_size = 0.5)
train = train.reset_index(drop = True)
valid = pd.concat([normal_valid, anormal_valid])
valid = valid.reset_index(drop = True)
test = pd.concat([normal_test, anormal_test])
test = test.reset_index(drop = True)
```

看一下各数据集形状：

```
train.shape    #  (227452, 31)
valid.shape    #  (28677, 31)
test.shape     #  (28678, 31)
#################################################################
#  Step 1:基于高斯分布的无监督异常值检测
##################################################### #########
'''计算协方差矩阵和均值'''
from scipy.stats import multivariate_normal
mu = train.drop('Class', axis = 1).mean(axis = 0).values
sigma = train.drop('Class', axis = 1).cov().values
```

```
#  多元正态分布,指定协方差、均值,允许奇异协方差矩阵
model = multivariate_normal(cov = sigma, mean = mu, allow_singular = True)
#  logpdf:概率密度函数对数
print(np.median(model.logpdf(valid[valid['Class'] == 0].drop('Class', axis = 1).values)))
print(np.median(model.logpdf(valid[valid['Class']  ==  1].drop('Class', axis = 1).values)))
```

运行结果如下：

```
-33.91079260533387
-716.0503164497757
```

这里解释一下上面代码的含义。

- multivariate_normal 是多元正态分布，指定协方差、均值，同时允许奇异协方差矩阵。mu 是训练集的均值，sigma 是训练集的协方差。
- logpdf 是计算概率密度函数的对数，这里打印出中位数，方便指定下面代码中的阈值范围。

```
'''计算不同阈值下的 recall_score、precision_score 和 fbeta_score'''
from sklearn.metrics import recall_score, precision_score, fbeta_score
tresholds = np.linspace(-1000, -10, 150)
scores = []
for treshold in tresholds:
    y_valid_lable = (model.logpdf(valid.drop('Class', axis = 1)) < treshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_lable), precision_score(valid
['Class'],
                 y_valid_lable),fbeta_score(valid['Class'], y_valid_lable, beta = 2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax())
```

这里解释一下各分数，recall_score 是计算召回率分数，precision_score 是计算命中率分数，fbeta_score 是精度和召回率的加权调和平均值。

运行结果如下：

```
0.7723250201126307  110
```

可视化不同阈值下的 recall_score、precision_score 和 fbeta_score，代码如下。

```
'''可视化不同阈值下的 recall_score、precision_score 和 fbeta_score'''
plt.plot(tresholds, scores[:, 0], label = '$Recall$')
plt.plot(tresholds, scores[:, 1], label = '$Precision$')
plt.plot(tresholds, scores[:, 2], label = '$F_2$')
```

```
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc = 'best')
plt.show()
```

可视化结果如图 4-4 所示。

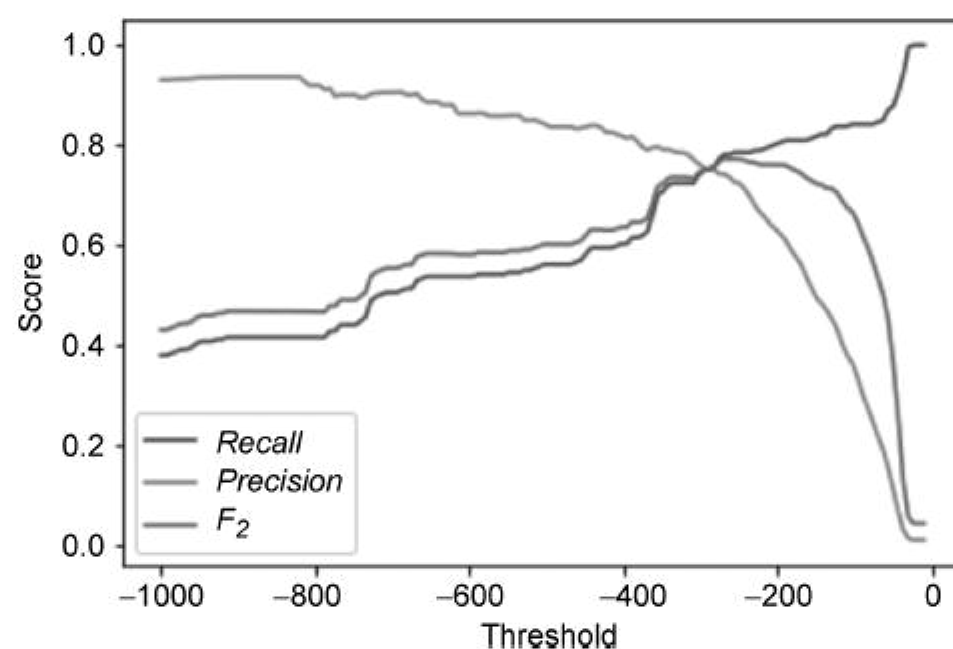


图 4-4 不同阈值下的 recall、precision、f2 可视化图

```
'''计算测试集评价标准'''
final_tresh = tresholds[scores[:, 2].argmax()]
y_test_label = (model.logpdf(test.drop('Class', axis = 1).values) < final_tresh).astype
(int)
print('最终阈值是: %d' % final_tresh)
print('测试集召回率分数是: %.3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率率分数是: %.3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是: %.3f' % fbeta_score(test['Class'], y_test_label, beta = 2))
```

运行结果如下：

```
最终阈值是: - 269
测试集召回率分数是:0.772
测试集命中率分数是:0.704
测试集 F2 分数是:0.758
```

混淆矩阵可视化,代码如下。

```
'''混淆矩阵可视化'''
import itertools
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(cm, classes,title = '混淆矩阵',cmap = plt.cm.Blues):
    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
```

```
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 0)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 3.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j], horizontalalignment = "center", color = "white" if cm[i, j] >
                thresh else "black")
    plt.tight_layout()
    plt.ylabel('实际')
    plt.xlabel('预测')
cnf_matrix = confusion_matrix(test['Class'], y_test_label)
class_names = [0,1]
plot_confusion_matrix(cnf_matrix, classes = class_names)
```

运行结果如图 4-5 所示。

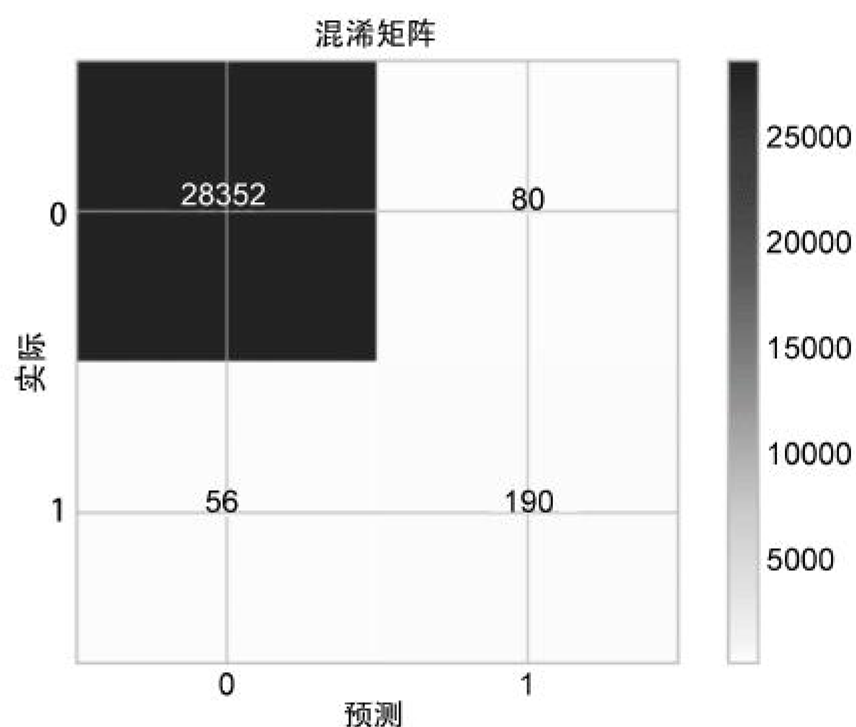


图 4-5　混淆矩阵图

整体来说，样本的召回率效果还是不错的，在这个极度不平衡的数据集中，找到了 190 个异常样本。

```
##########################################################
#   Step 2:基于混合高斯模型的无监督异常值检测
##########################################################
'''混合高斯模型'''
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components = 3, n_init = 4, random_state = 1)
gmm.fit(train.drop('Class', axis = 1).values)
print(gmm.score(valid[valid['Class'] == 0].drop('Class', axis = 1).values))
print(gmm.score(valid[valid['Class'] == 1].drop('Class', axis = 1).values))
```

运行结果如下：

```
6.8138687337843695
-132514.22047425093
```

下面的方法、原理与高斯模型一致。

```
'''计算不同阈值下的 recall_score、precision_score 和 fbeta_score'''
tresholds = np.linspace( - 140000, 20, 1000)
scores = []
y_pred_score = gmm.score_samples(valid.drop('Class', axis=1).values)
for treshold in tresholds:
    y_valid_lable = (y_pred_score < treshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_lable), precision_score(valid
['Class'],
                    y_valid_lable), fbeta_score(valid['Class'], y_valid_lable, beta=2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax())  #fbeta 最大为 0.81,索引是 68
'''可视化不同阈值下的 recall_score、precision_score 和 fbeta_score'''
plt.plot(tresholds, scores[:, 0], label='$Recall$')
plt.plot(tresholds, scores[:, 1], label='$Precision$')
plt.plot(tresholds, scores[:, 2], label='$F_2$')
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc='best')
plt.show()
'''计算测试集评价标准'''
final_tresh = tresholds[scores[:, 2].argmax()]
y_test_label=(gmm.score_samples(test.drop('Class', axis=1).values) < final_tresh).
astype(int)
print('最终阈值是:%d' % final_tresh)
print('测试集召回率分数是:%.3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率率分数是:%.3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是:%.3f' % fbeta_score(test['Class'], y_test_label, beta=2))
cnf_matrix = confusion_matrix(test['Class'].values, y_test_label)
plot_confusion_matrix(cnf_matrix, classes=class_names)
```

最终运行结果如图 4-6 所示。

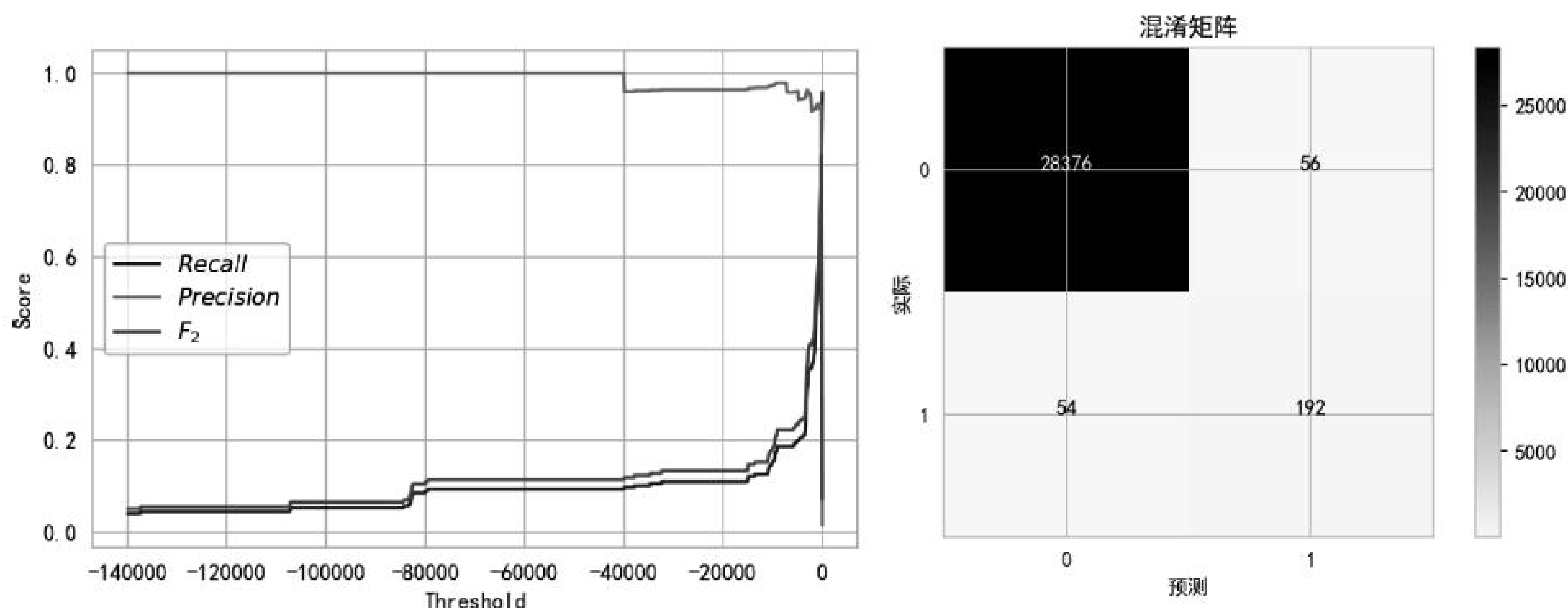


图 4-6 混合高斯模型下的相关结果展示图

下面再介绍孤立森林和直方图的异常值检测方法，代码如下：

```
##########################################################
#   Step 3:基于孤立森林的无监督异常值检测
##########################################################
'''孤立森林模型'''
from sklearn.ensemble import IsolationForest
model = IsolationForest(n_estimators = 50, bootstrap = True, max_samples = train.shape[0],
                        n_jobs = 4, random_state = 1)
model.fit(train.drop('Class', axis = 1).values)
print(model.decision_function(valid[valid['Class'] == 0].drop('Class', axis = 1).values).
mean())
print(model.decision_function(valid[valid['Class'] == 1].drop('Class', axis = 1).values).
mean())
'''计算不同阈值下的 recall_score、precision_score 和 fbeta_score'''
tresholds = np.linspace( - 0.2, 0.2, 200)
scores = []
y_pred_score = model.decision_function(valid.drop('Class', axis = 1).values)
for treshold in tresholds:
    y_valid_lable = (y_pred_score < treshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_lable), precision_score(valid['Class'],
                  y_valid_lable), fbeta_score(valid['Class'], y_valid_lable, beta = 2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax())
'''可视化不同阈值下的 recall_score、precision_score 和 fbeta_score'''
plt.plot(tresholds, scores[:, 0], label = '$ Recall $ ')
plt.plot(tresholds, scores[:, 1], label = '$ Precision $ ')
plt.plot(tresholds, scores[:, 2], label = '$ F_2 $ ')
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc = 'best')
plt.show()
'''计算测试集评价标准'''
final_tresh = tresholds[scores[:, 2].argmax()]
y_test_label = (model.decision_function(test.drop('Class',axis = 1).values)< final_tresh).
                astype(int)
print('最终阈值是: % d' % final_tresh)
print('测试集召回率分数是: % .3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率率分数是: % .3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是: % .3f' % fbeta_score(test['Class'], y_test_label, beta = 2))
cnf_matrix = confusion_matrix(test['Class'].values, y_test_label)
plot_confusion_matrix(cnf_matrix, classes = class_names)
```

运行结果如图 4-7 所示。

```
##########################################################
#   Step 4:基于直方图的异常值检测
##########################################################
```

```
'''直方图探索:摆脱了高斯分布的限制,但是没有捕捉到变量之间的关系'''
class hist_model(object):
    def __init__(self, bins = 50):
        self.bins = bins
    def fit(self, X):
        bin_hight, bin_edge = [], []
        for var in X.T:
            bh, bedge = np.histogram(var, bins = self.bins)
            bin_hight.append(bh)
            bin_edge.append(bedge)
        self.bin_hight = np.array(bin_hight)
        self.bin_edge = np.array(bin_edge)
    def predict(self, X):
        scores = []
        for obs in X:
            obs_score = []
            for i, var in enumerate(obs):
                bin_num = (var > self.bin_edge[i]).argmin() - 1
                obs_score.append(self.bin_hight[i, bin_num])
                scores.append(np.mean(obs_score))
        return np.array(scores)
model = hist_model()
#  为每个特征建立直方图,将 bin 的高度合并为分数,如果分数很低,说明异常
model.fit(train.drop('Class', axis = 1).values)
print(np.median(model.predict(valid[valid['Class'] == 0].drop('Class', axis = 1).
values)))
print(np.median(model.predict(valid[valid['Class'] == 1].drop('Class', axis = 1).
values)))
'''计算不同阈值下的 recall_score、precision_score 和 fbeta_score'''
tresholds = np.linspace(10000, 80000, 100)
scores = []
y_pred_score = model.predict(valid.drop('Class', axis = 1).values)
for treshold in tresholds:
    y_valid_lable = (y_pred_score < treshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_lable), precision_score(valid['Class'],
                   y_valid_lable), fbeta_score(valid['Class'], y_valid_lable, beta = 2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax())  # fbeta 最大为 0.42,索引是 42
'''可视化不同阈值下的 recall_score、precision_score 和 fbeta_score'''
plt.plot(tresholds, scores[:, 0], label = '$Recall$')
plt.plot(tresholds, scores[:, 1], label = '$Precision$')
plt.plot(tresholds, scores[:, 2], label = '$F_2$')
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc = 'best')
plt.show()
'''计算测试集评价标准'''
```

```
final_tresh = tresholds[scores[:, 2].argmax()]
y_test_label = (model.predict(test.drop('Class', axis = 1).values) < final_tresh).astype(int)
print('最终阈值是：%d' % final_tresh)
print('测试集召回率分数是：%.3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率率分数是：%.3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是：%.3f' % fbeta_score(test['Class'], y_test_label, beta = 2))
cnf_matrix = confusion_matrix(test['Class'].values, y_test_label)
plot_confusion_matrix(cnf_matrix, classes = class_names)
```

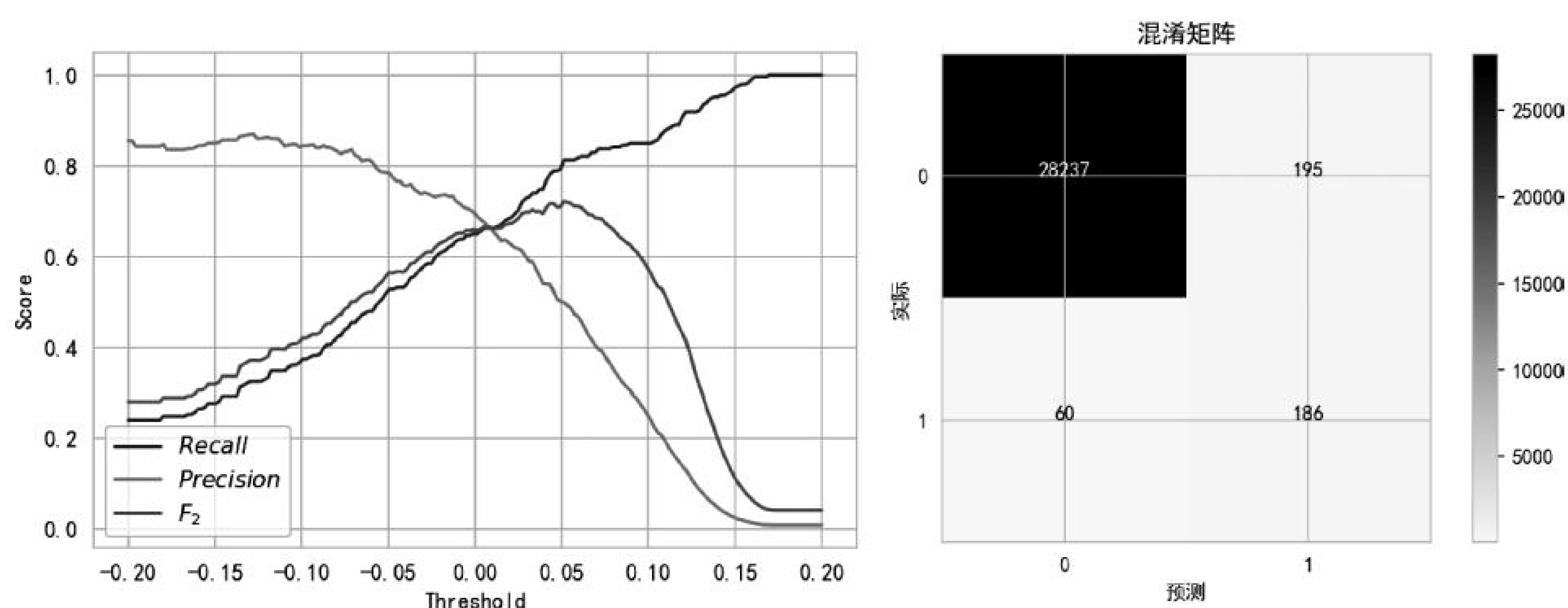


图 4-7　孤立森林模型下的相关结果展示图

运行结果如图 4-8 所示。

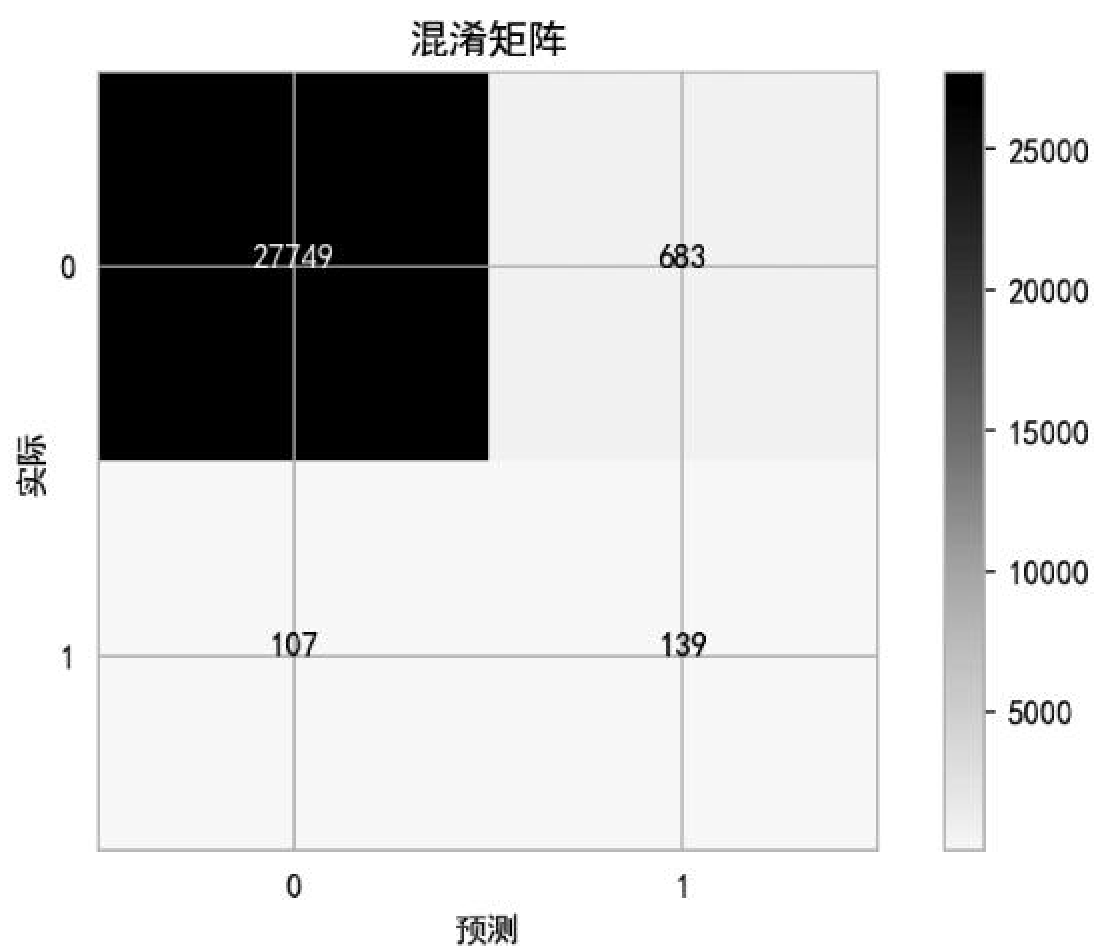


图 4-8　直方图异常检测模型下的相关结果展示图

最后介绍基于 SVM 的异常值检测。

```
##########################################################
#   Step 5:基于 SVM 的异常值检测
##########################################################
'''一类 svm,用于异常值检测'''
from sklearn.svm import OneClassSVM
model = OneClassSVM()
model.fit(train.drop('Class', axis = 1).values)
print(model.decision_function(valid[valid['Class'] == 0].drop('Class', axis = 1).values).
mean())
print(model.decision_function(valid[valid['Class'] == 1].drop('Class', axis = 1).values).
mean())
'''计算不同阈值下的 recall_score、precision_score 和 fbeta_score'''
tresholds = np.linspace( - 50000, - 400, 500)
scores = []
y_pred_score = model.decision_function(valid.drop('Class', axis = 1).values)
for treshold in tresholds:
    y_valid_lable = (y_pred_score < treshold).astype(int)
    scores.append([recall_score(valid['Class'], y_valid_lable), precision_score(valid['Class'],
                   y_valid_lable), fbeta_score(valid['Class'], y_valid_lable, beta = 2)])
scores = np.array(scores)
print(scores[:, 2].max(), scores[:, 2].argmax())  # fbeta 最大为 0.81,索引是 68
'''可视化不同阈值下的 recall_score、precision_score 和 fbeta_score'''
plt.plot(tresholds, scores[:, 0], label = '$ Recall $ ')
plt.plot(tresholds, scores[:, 1], label = '$ Precision $ ')
plt.plot(tresholds, scores[:, 2], label = '$ F_2 $ ')
plt.ylabel('Score')
plt.xlabel('Threshold')
plt.legend(loc = 'best')
plt.show()
'''计算测试集评价标准'''
final_tresh = tresholds[scores[:, 2].argmax()]
y_test_label = (model.decision_function(test.drop('Class',axis = 1).values)< final_tresh).
               astype(int)
print('最终阈值是: % d' % final_tresh)
print('测试集召回率分数是: % .3f' % recall_score(test['Class'], y_test_label))
print('测试集命中率率分数是: % .3f' % precision_score(test['Class'], y_test_label))
print('测试集 F2 分数是: % .3f' % fbeta_score(test['Class'], y_test_label, beta = 2))
cnf_matrix = confusion_matrix(test['Class'].values, y_test_label)
plot_confusion_matrix(cnf_matrix, classes = class_names)
```

SVM 最大的缺陷是在大数据集上计算的过程较为缓慢,所以建议慎用。

4.3　小结

在日常工作中，会经常碰到不平衡样本。一般来说，建立模型所需要的坏样本比例最好在20%以上。如果坏样本比例1%以上，可以考虑过采样/欠采样；如果坏样本比例1%以下，可以考虑以上基于分布的异常值检测方法。但要注意以下两点。

（1）数据质量决定模型效果。数据质量不好的话，建模效果也不会太理想。

（2）不平衡样本哪怕再怎么用技巧，有时候模型效果也会很不理想。模型不是万能的，也不会有某个算法是万能的，最应该做的是将模型与具体业务进行结合，使模型真正发挥实际作用。

第 5 章



自然语言处理案例——电商评论



视频讲解

自然语言处理通常用于分词、搜索、模糊匹配、正则表达式、词云、情感分析；深一些来说，自然语言处理一般与语音识别结合在一起，用深度学习神经网络训练，结合语料库，进行语义分析、人机问答等。

本章主要通过电商评论案例来讲解自然语言的实际应用。

5.1 数据加载与预处理

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data/电商评论')
df = pd.read_csv("comment.csv")
df.head()
```

数据预览结果如图 5-1 所示。

```
In [3]: df.head()
Out[3]:
```

	Unnamed: 0	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Department Name	Class Name
0	0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initmates	Intimate	Intimates
1	1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses	Dresses
2	2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses	Dresses
3	3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms	Pants
4	4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops	Blouses

图 5-1 电商评论数据预览展示图

由图 5-1 可以发现第 1 列也是序号列，与数据自动创建的索引重复，所以删除第 1 列。

```
df.drop(df.columns[0], inplace = True, axis = 1)              #删除第 1 列
```

该数据集各特征列是英文，在这里使用 rename()函数将列名重命名为中文名。

```
new_col = {'Clothing ID':'服装 id', 'Age':'评论者年龄', 'Title':'评论标题',
           'Review Text':'评论内容', 'Rating':'服装评分', 'Recommended IND':'是否推荐',
           'Positive Feedback Count':'赞同该评论的人数', 'Division Name':'产品高级分类',
           'Department Name':'产品大类', 'Class Name':'产品二级分类'}
df = df.rename(columns = new_col)                    #重命名
df.head()
```

再次预览数据，如图 5-2 所示。

```
In [2]: df.head()
Out[2]:
```

	服装id	评论者年龄	评论标题	评论内容	服装评分	是否推荐	赞同该评论的人数	产品高级分类	产品大类	产品二级分类
0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initmates	Intimate	Intimates
1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses	Dresses
2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses	Dresses
3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms	Pants
4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops	Blouses

图 5-2 电商评论数据重命名后的预览展示图

使用 missingno 包预览缺失值。

```
import missingno
missingno.matrix(df,fontsize = 25)                  #缺失值可视化
```

缺失值可视化结果如图 5-3 所示。

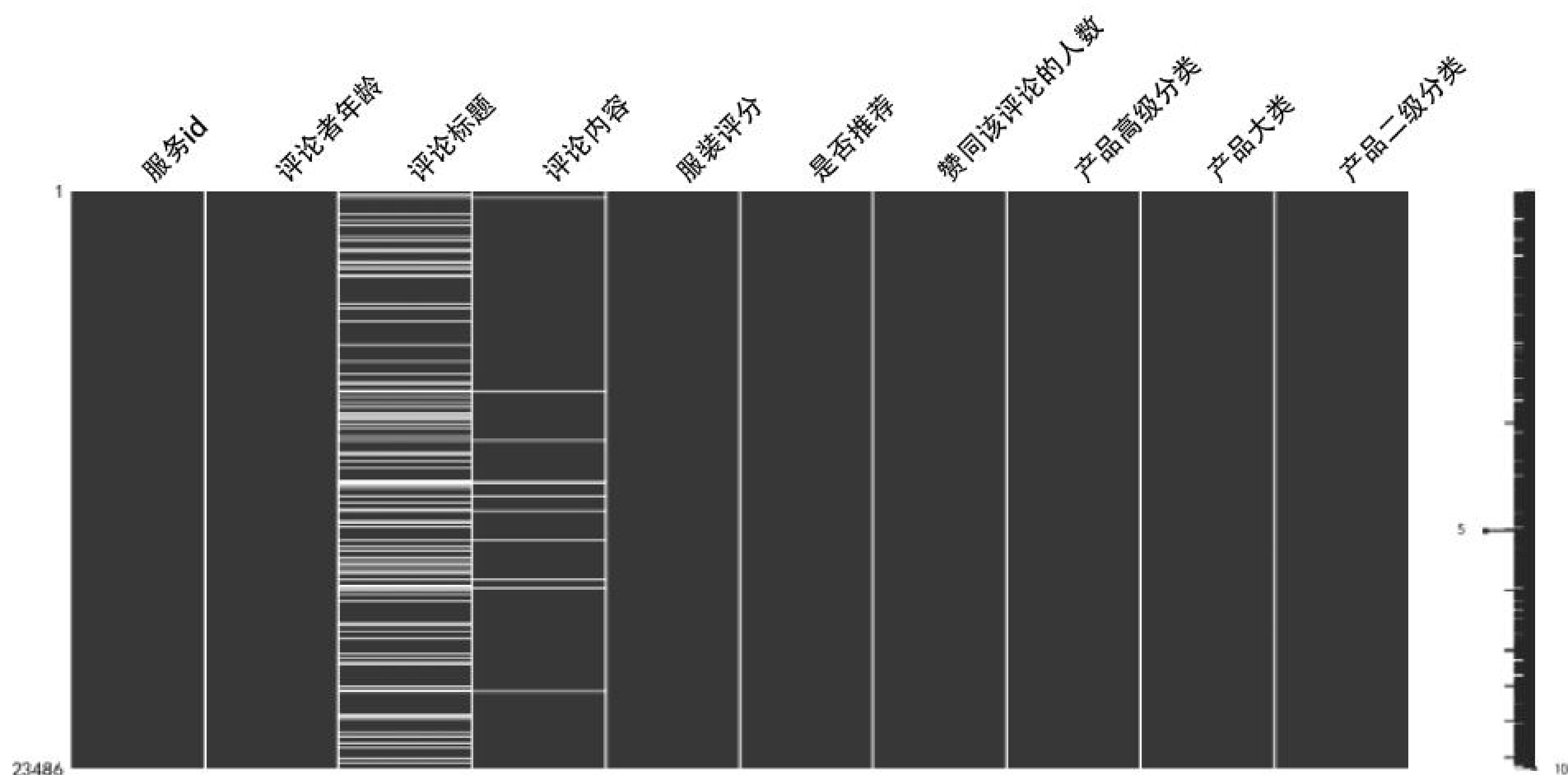


图 5-3 数据缺失值可视化图

由图 5-3 可以发现评论内容有缺失值,那么删除缺失值。

```
col = ["评论内容"]
df = df.dropna(subset = [col])                 # 删除缺失值
```

可以做一些特征衍生的工作,在这里创建了新特征:评论字数。

```
df['评论内容'] = df['评论内容'].astype(str)
df['评论字数'] = df['评论内容'].apply(len)            # 计算字符串总长度
```

5.2 数据可视化

为了查看不同变量与评分之间的关系,可以使用可视化进行展示。

```
'''评分与字数关系的可视化 1'''
g = sns.FacetGrid(data = df, col = '服装评分')
g.map(plt.hist, '评论字数', bins = 50)
plt.show()
```

结果如图 5-4 所示。

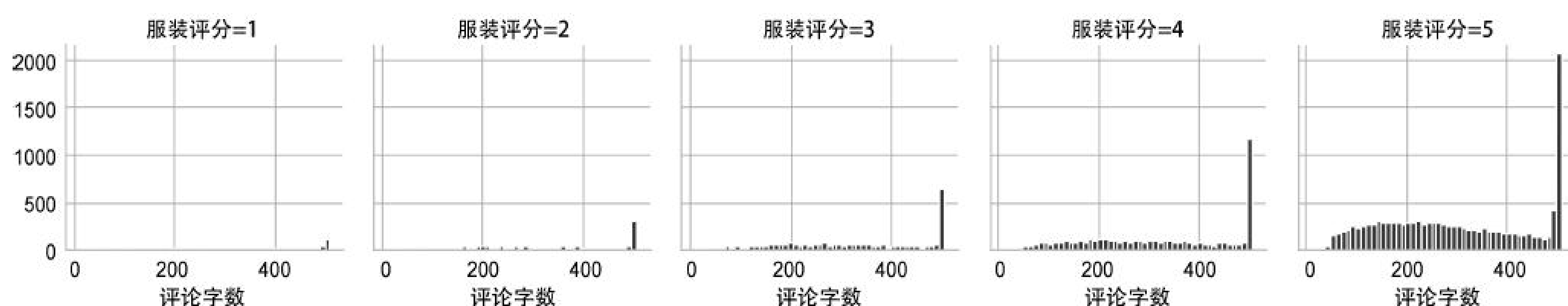


图 5-4 服装评分与评论字数的可视化图(1)

```
'''评分与字数关系的可视化 2'''
sns.pointplot(x = '服装评分', y = '评论字数', data = df)
plt.show()
```

结果如图 5-5 所示。

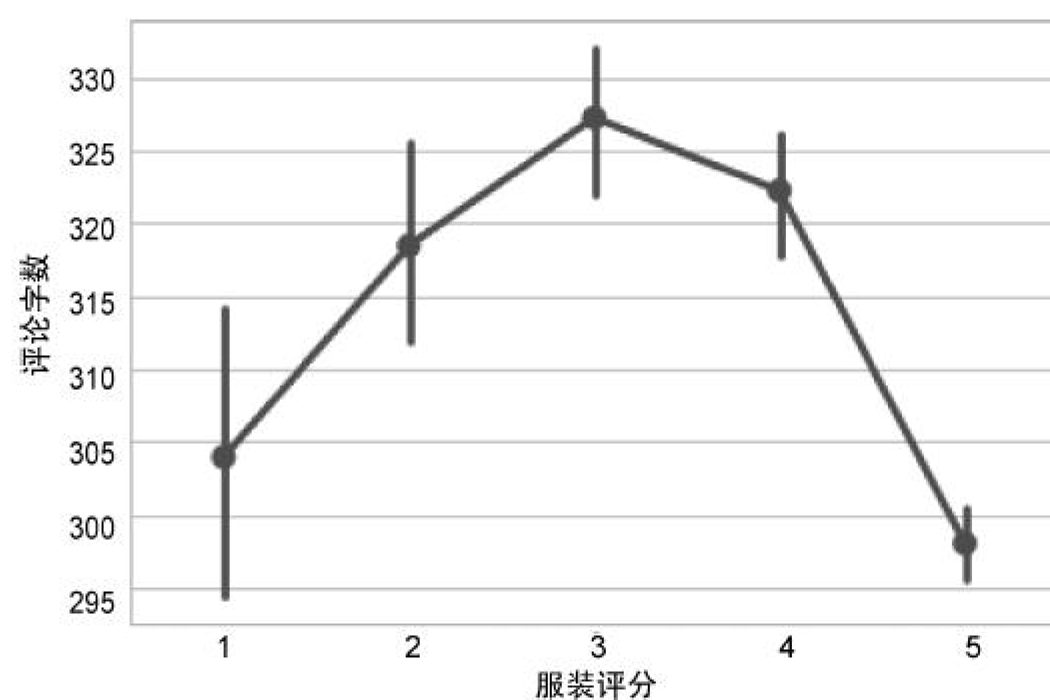


图 5-5 服装评分与评论字数的可视化图(2)

由图 5-4 可以发现,评分越高,写评论的人就越多。由图 5-5 可以发现,就平均字数而言,评分为 3 的人最喜欢写长评论。

```
'''找出与服装评分相关性最强的 10 个变量'''
k = 10
corr = round(df.corr(), 2)
cols = corr.nlargest(k, '服装评分')['服装评分'].index
cm = round(df[cols].corr(), 2)
mask = np.zeros_like(cm, dtype = np.bool)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize = (8, 8))
cmap = sns.diverging_palette(220, 10, as_cmap = True)
sns.heatmap(cm, mask = mask, cmap = cmap, center = 0, annot = True, cbar_kws = {"shrink": .5})
```

运行结果如图 5-6 所示。

由图 5-6 可以发现,“是否推荐”这个变量与“服装评分”相关性最强。

```
'''服装评分与年龄的关系'''
df.groupby(['服装评分', pd.cut(df['评论者年龄'], np.arange(0,100,10))]).size().unstack(0).
              plot.bar(stacked = True)
plt.show()
```

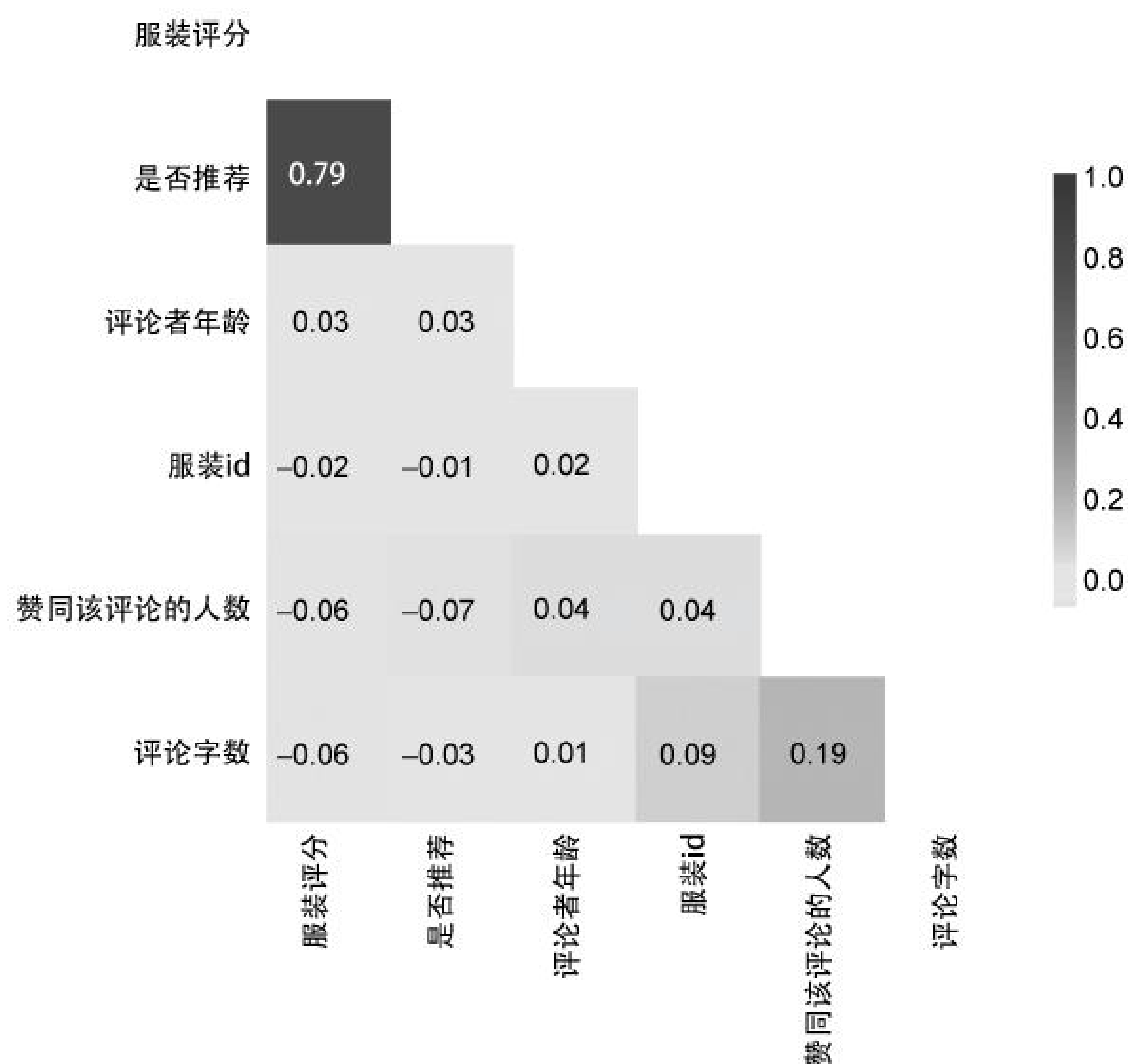


图 5-6　与服装评分相关性最强的 10 个变量展示图

运行结果如图 5-7 所示。

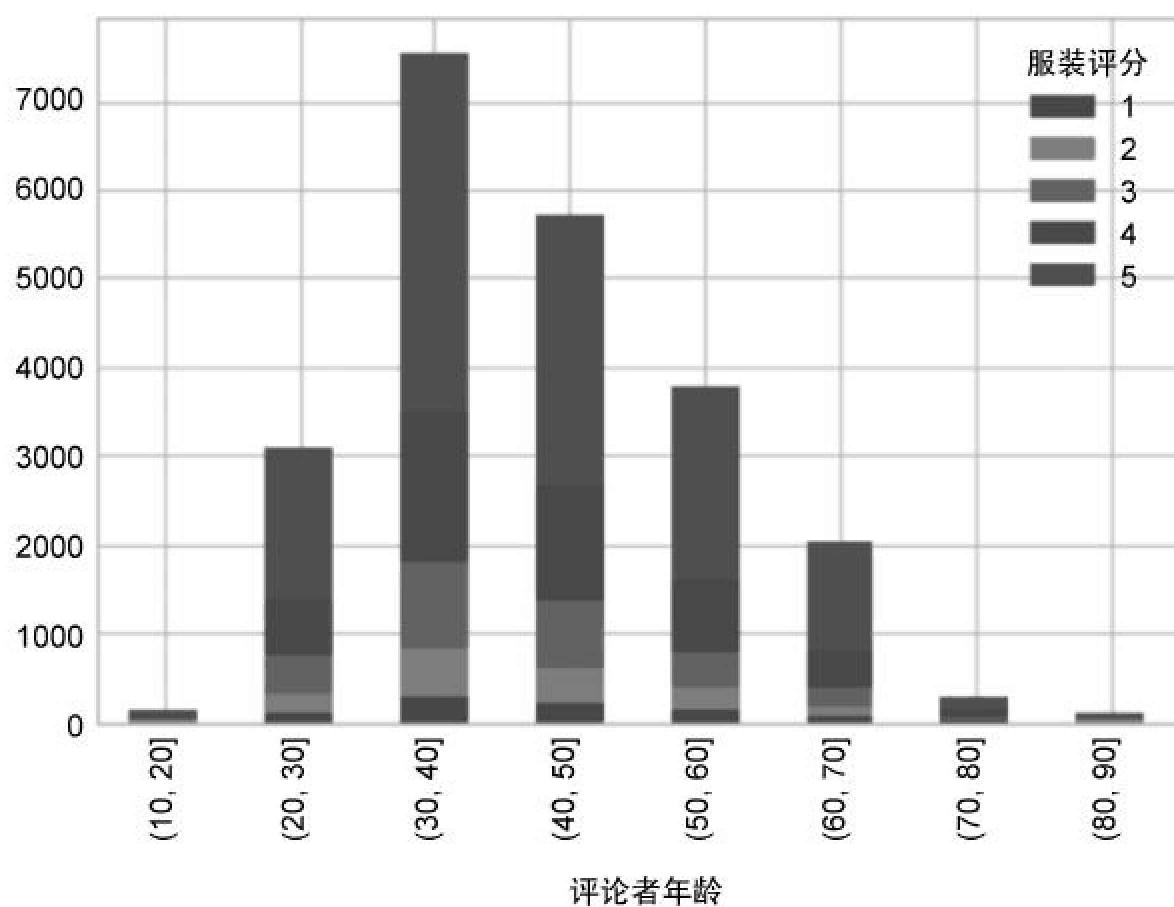


图 5-7　服装评分与评论者年龄的关系展示图

由图 5-7 可以发现,“30～40”与“40～50”这两个年龄段的人在评论时比较喜欢给高分。

```
'''产品类别与年龄的关系'''
df.groupby(['产品大类', pd.cut(df['评论者年龄'], np.arange(0,100,10))]).size().unstack(0).
            plot.bar(stacked = True)
plt.show()
```

运行结果如图 5-8 所示。

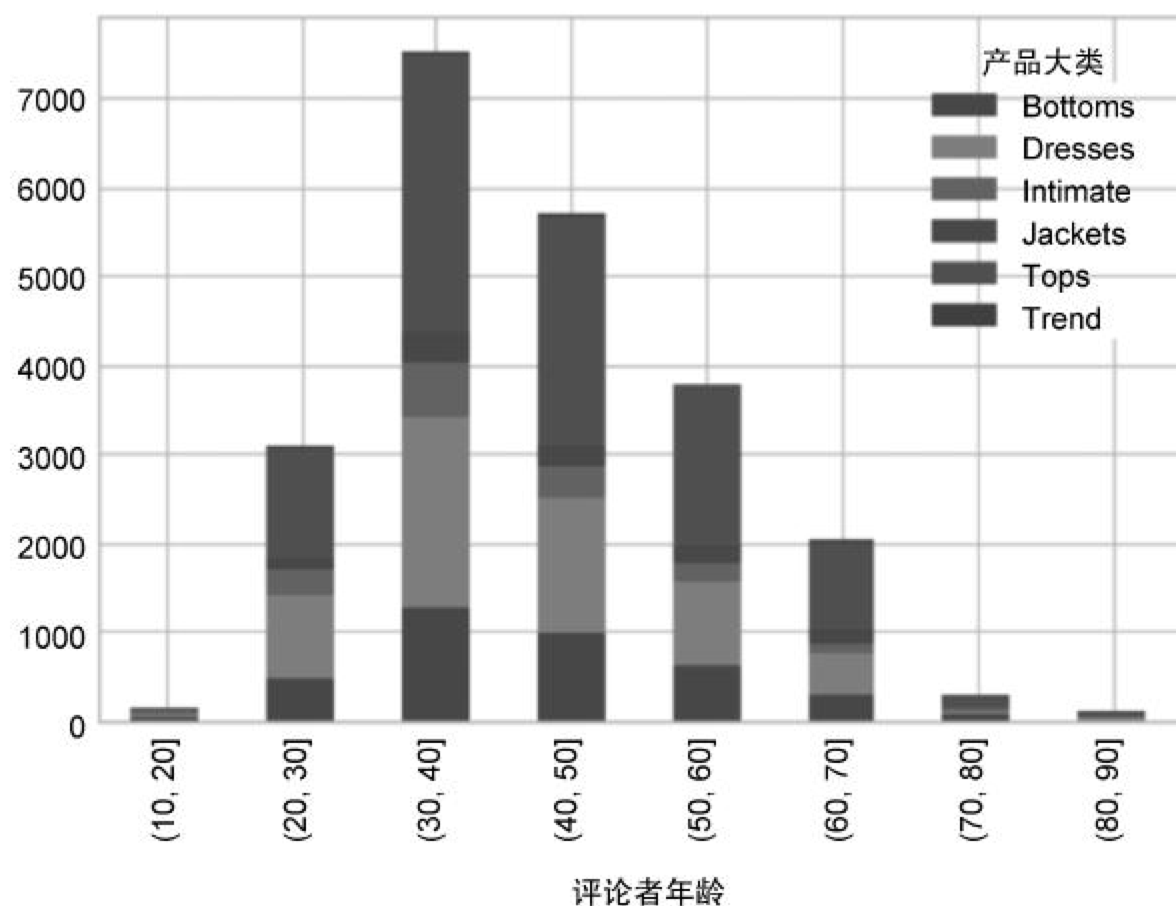


图 5-8　产品大类与评论者年龄的关系展示图

由图 5-8 可以发现,"30～40"和"40～50"是购物的主要年龄段;在各产品类别中,"Tops"在各年龄段的销量都不错,"Dresses"和"Bottoms"在"30～40"销量最高。

5.3　文本分析

对于电商评论内容,除了衍生出评论字数这一变量外,还可对其进行文本分析,找出评论内容中最常见的词语有哪些,然后再对常用词进行情感分析。

```
'''字符串清洗'''
import re
a = df['评论内容'].str.lower().str.cat(sep = ' ')
b = re.sub('[^A - Za - z] + ', ' ', a)
'''加载停用词库,从字符串文本中移除停用词'''
from stop_words import get_stop_words
stop_words = list(get_stop_words('english'))                    #选择英语停用词
import nltk
from nltk.corpus import stopwords
nltk_words = list(stopwords.words('english'))
```

```
stop_words.extend(nltk_words)
from nltk import word_tokenize
word_tokens = word_tokenize(b)               #分词
filtered_sentence = [i for i in word_tokens if i not in stop_words]
```

在这里说明一下,不同国家的停用词库是不一样的。如果文本内容是中文,可以在网上下载中文停用词库;如果文本内容是英语,Python 有专门打包好的停用词库。

这里使用了两个英语停用词库,一个是 stop_words. get_stop_words,可以打开网址:https://pypi.org/project/stopwords/,查看支持的国家语言;一个是有名的自然语言处理包 nltk,在用这个包之前,需要在 pycharm 中输入 nltk. download()语句,下载停用词包。

停用词库准备好后,就是分词和去除停用词了。如果文本内容是中文,一般用 jieba 分词,如果文本内容是英文,一般用 nltk 里的 word_tokenize。

分词过后,一般需要删除过短的评论。

```
'''选择长度大于 2 的字符'''
without_single_chr = [i for i in filtered_sentence if len(i) > 2]
'''计算词频'''
top_N = 100
word_dist = nltk.FreqDist(without_single_chr)
rslt = pd.DataFrame(word_dist.most_common(top_N), columns = ['单词', '频率'])
sns.barplot(x = '单词', y = '频率', data = rslt.head(7))
plt.show()
```

运行结果如图 5-9 所示。

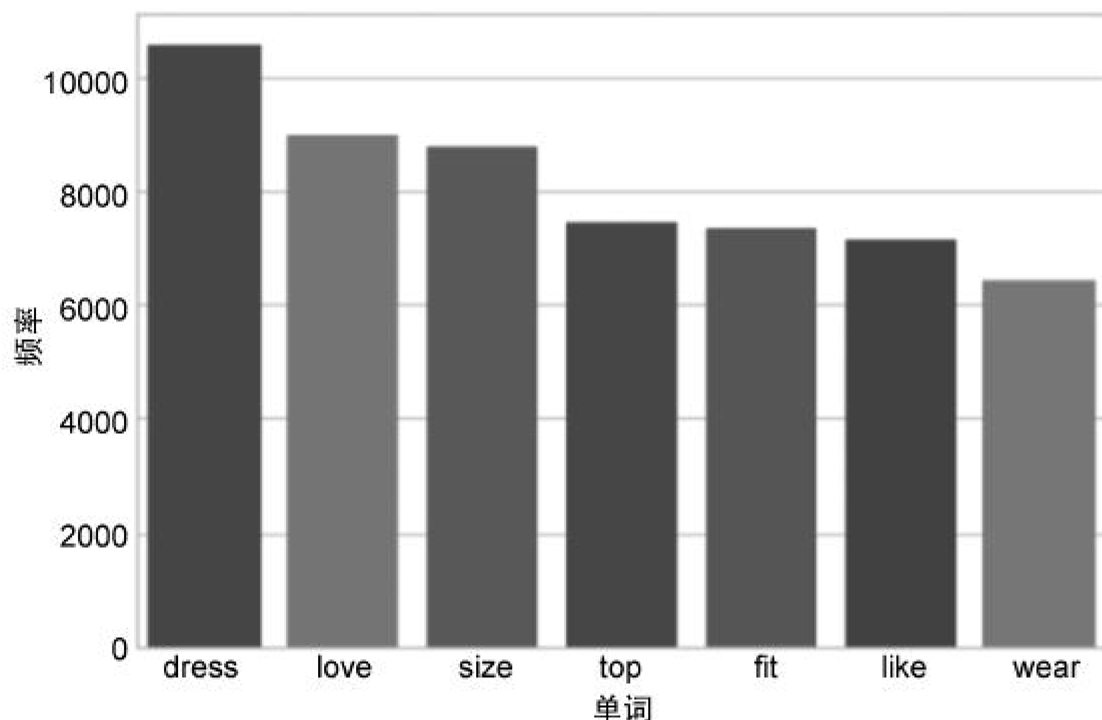


图 5-9　不同单词的词频统计展示图

```
'''词云图'''
from wordcloud import WordCloud
def wc(data, bgcolor, title):
    plt.figure(figsize = (10,10))
    plt.title(title)
    wc = WordCloud(background_color = bgcolor, max_words = 1000, max_font_size = 50)
    wc.generate(' '.join(data))
    plt.imshow(wc)
    plt.axis('off')
    plt.show()
wc(without_single_chr, 'black', '最常用的单词')
```

运行结果如图 5-10 所示。

图 5-10 最常用的单词词云图

5.4 情感分析

情感分析是对带有情感色彩的主观性文本进行分析、处理、归纳和推理的过程。在电商的评论内容中，商家可以根据购买者评论的内容，分析出购买者对商品的喜欢程度，从而对商品或服务进行优化与改进。

```
from textblob import TextBlob                        #情感分析
bloblist_desc = []
df_review_str = df['评论内容'].astype(str)
for i in df_review_str:
    blob = TextBlob(i)
    #polarity 正数表示积极，负数表示消极，subjectivity 0 表示客观，1 表示主观
    bloblist_desc.append((i, blob.sentiment.polarity, blob.sentiment.subjectivity))
    df_polarity_desc = pd.DataFrame(bloblist_desc, columns = ['评论', '情绪', '极性'])
def f(x):
    if x['情绪'] > 0:
```

```
        val = '积极评论'
    elif x['情绪'] == 0:
        val = '中性评论'
    else:
        val = '负面评论'
    return val
df_polarity_desc['情绪类型'] = df_polarity_desc.apply(f, axis=1)
sns.countplot(x='情绪类型', data=df_polarity_desc)
plt.show()
```

运行结果如图 5-11 所示。

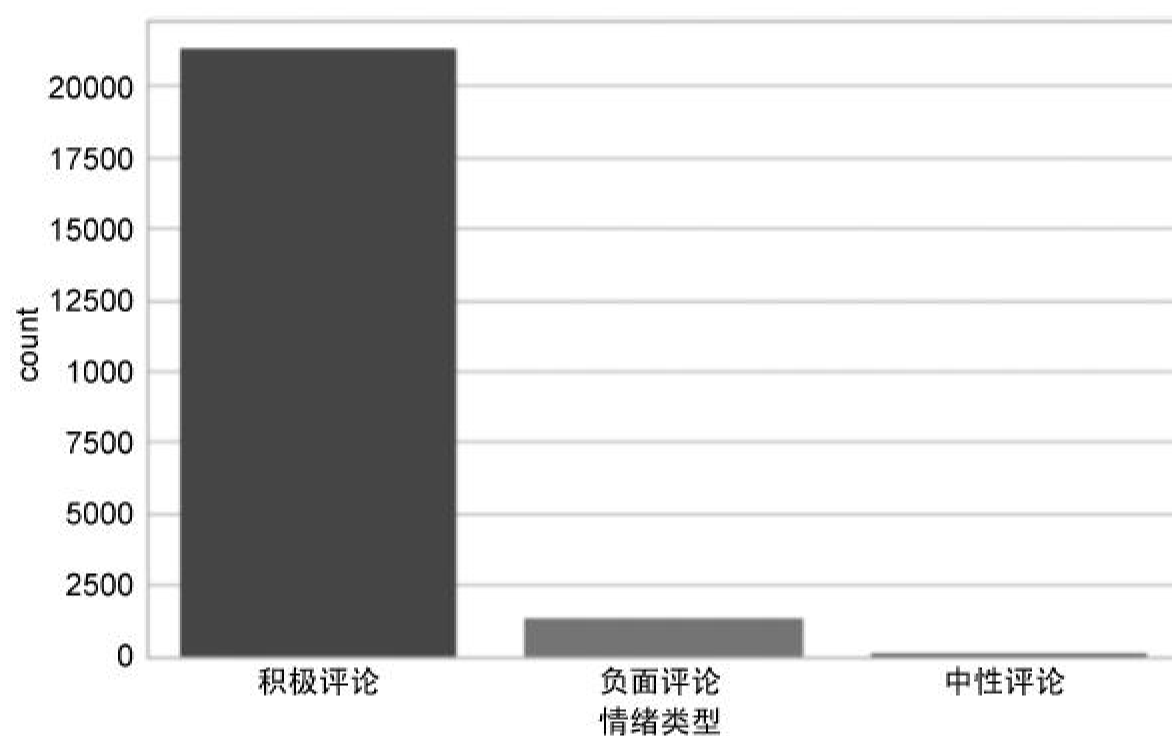


图 5-11　情感分析词频图

由图 5-11 可以发现,绝大部分评论还是积极的。

```
'''正面评论词云图'''
positive_reviews = df_polarity_desc[df_polarity_desc['情绪类型'] == '积极评论']
wc(positive_reviews['评论'], 'black', '正面评论')
```

运行结果如图 5-12 所示。



图 5-12　正面评论词云图

```
'''负面评论词云图'''
negative_reviews = df_polarity_desc[df_polarity_desc['情绪类型'] == '负面评论']
wc(negative_reviews['评论'], 'black', '负面评论')
```

运行结果如图 5-13 所示。

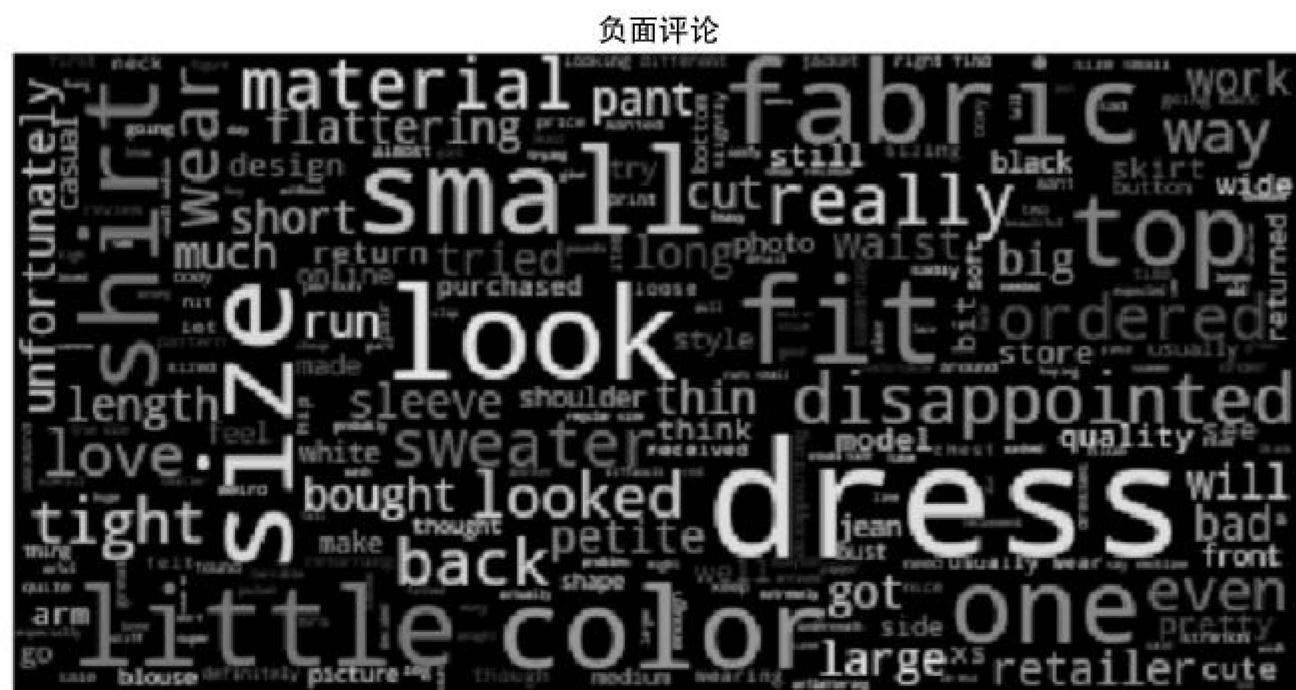


图 5-13　负面评论词云图

5.5　文本分类

在对电商评论内容的常用词语进行情感分析之后，可根据顾客对商品的喜爱程度，推荐其他类似的商品，从而提高商品销量和商家的收益。

```
'''停用词处理'''
import string
def text_process(x):
    nopunc = [i for i in x if i not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [i for i in nopunc.split() if i.lower() not in stopwords.words('english')]
df['评论内容'] = df['评论内容'].apply(text_process)
'''朴素贝叶斯模型进行文本分类'''
rating_class = df[(df['服装评分'] == 1) | (df['服装评分'] == 5)]
X_review = rating_class['评论内容']
y = rating_class['服装评分']
from sklearn.feature_extraction.text import CountVectorizer #文本特征提取
bow_transformer = CountVectorizer(analyzer = text_process).fit(X_review)
X_review = bow_transformer.transform(X_review)
from sklearn.model_selection import train_test_split as sp
X_train,X_test,y_train,y_test = sp(X_review, y, test_size = 0.3, random_state = 101)
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
```

```
nb.fit(X_train, y_train)
predict = nb.predict(X_test)
'''预测推荐产品'''
X_predict_recommend = df['评论内容']
y_recommend = df['是否推荐']
bow_transformer = CountVectorizer(analyzer = text_process).fit(X_predict_recommend)
X_predict_recommend = bow_transformer.transform(X_predict_recommend)
X_train, X_test, y_train, y_test = sp(X_predict_recommend, y_recommend,
                test_size = 0.3, random_state = 101)
nb = MultinomialNB()
nb.fit(X_train, y_train)
predict_recommendation = nb.predict(X_test)
```

5.6 小结

本章主要通过电商评论案例,简单地介绍了自然语言处理的流程,其中包括可视化、文本分析、情感分析以及文本分类等。现阶段,自然语言处理也是人工智能的重要应用方向之一,感兴趣的读者可以先确定好自己的研究方向,再进行深入研究。

第 6 章

模型融合

模型融合是综合考虑不同模型的情况，然后再将它们的输出结果融合到一起。模型融合主要有以下 5 种方法：bagging、boosting、stacking、blending 和输出结果加权融合。

为什么要做模型融合？因为它通常可以使模型效果获得提升，同时，模型融合是机器学习竞赛中常用的方法。

本章主要介绍 stacking 和 blending 方法。

1. 两层 *N* 个模型的 *K* 折 stacking 原理

(1) 将模型列表中每个模型的输出结果作为新的训练集特征，用于第 2 层模型。

(2) 单个模型 *K* 折交叉训练，将每折训练出的模型放在测试集上预测，输出 *K* 次结果，均值作为新的测试集特征，用于第 2 层模型。

(3) 把 New Features 和 label 作为新的分类器的输入进行训练，然后通过测试集的 New Features 获得最终的预测结果。

2. 两层 *N* 个模型的 blending 原理

(1) 训练集划分为两部分(train1，train2)，测试集为 test。

(2) 模型列表中,每个模型在训练集 train2 部分的预测结果作为 X,与 train2 部分的 y 一起,组成新的训练集,用于第 2 层模型。

(3) 模型列表中,每个模型在测试集上的预测结果作为 X,组成新的测试集,用于第 2 层模型。

【注意】

基模型之间的差异性要尽可能大,最好是融合不同类型的模型;基模型之间的性能表现不能差距太大,如果有性能特别不好的模型,建议从模型列表中剔除。

6.1 分类模型的融合方法



视频讲解

用于风控领域的机器学习建模主要包括两大类:分类模型与回归模型。下面介绍分类模型的融合方法。

首先是导入包和加载数据,代码如下:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data')
data = pd.read_csv('data1.csv')
data.head()
```

预览数据,如图 6-1 所示。

```
In [2]: data.head()
Out[2]:
```

	贷款金额	贷款期限	利率	每月还款金额	贷款等级	工作年限	房屋所有权	年收入	月负债比	过去两年借款人逾期30天以上的数字	...	贷款目的_MOVING	贷款目的_OTHER	贷款目的_RENEWABLE_ENERGY	贷款目的_SMALL_BUSINESS
0	0.417669	1.0	-1.142293	0.465600	1.0	2.0	2.0	-0.268487	1.032490	-0.381014	...	0.0	0.0	0.0	0.0
1	-0.124930	1.0	-1.680677	-0.150988	1.0	10.0	2.0	0.272756	0.011931	-0.381014	...	0.0	0.0	0.0	0.0
2	-0.993088	1.0	-0.638188	-0.969661	2.0	10.0	3.0	-0.074261	-0.668442	-0.381014	...	0.0	1.0	0.0	0.0
3	-0.331118	1.0	-1.680677	-0.355540	1.0	3.0	3.0	-0.268487	0.011931	-0.381014	...	0.0	0.0	0.0	0.0
4	1.177307	2.0	3.031694	1.269438	6.0	10.0	3.0	0.039781	-0.482416	-0.381014	...	0.0	0.0	0.0	0.0

5 rows × 72 columns

图 6-1 数据预览展示图

看一下目标变量的水平分布。

```
data['target'].value_counts()
0     7963
1     2037
Name: target, dtype: int64
```

然后对数据集进行切分,代码如下:

```
allFeatures = list(data.columns)
allFeatures.remove('target')
X = data[allFeatures]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
```

用默认参数在训练集上训练模型。

```
from sklearn.linear_model import LogisticRegression as LR
lr = LR(random_state=1)
lr.fit(X_train, y_train)
from sklearn import metrics
y_test_label = lr.predict(X_test)
y_test_value = lr.predict_proba(X_test)[:, 1]
print("测试集准确率是: {:.2%}".format(metrics.accuracy_score(y_test, y_test_label)))
print("测试集 AUC 是: {:.4}".format(metrics.roc_auc_score(y_test, y_test_value)))
```

输出结果如下:

```
测试集准确率是: 80.50%
测试集 AUC 是: 0.6815
```

下面使用5折交叉训练试试,代码如下:

```
from sklearn.cross_validation import KFold
kf = KFold(len(y_train), 5, random_state=1)
scores = []
for iteration, indices in enumerate(kf, start=1):
    X_train = X_train.reset_index(drop=True)
    y_train = y_train.reset_index(drop=True)
    lr = LR(random_state=1)
    lr.fit(X_train.iloc[indices[0],:],y_train.iloc[indices[0]].values.ravel())
    y_pred = lr.predict_proba(X_train.iloc[indices[1],:].values)[:, 1]
    score = metrics.roc_auc_score(y_train.iloc[indices[1]].values, y_pred)
    scores.append(score)
```

```
    print('迭代次数: ', iteration,': 得分 = ', score)
print('平均得分 ', np.mean(scores))
y_test_value = lr.predict_proba(X_test)[:, 1]
print('测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, y_test_value)))
```

输出结果如下：

```
迭代次数 1: 得分 = 0.691516182342
迭代次数 2: 得分 = 0.666559048428
迭代次数 3: 得分 = 0.666094545455
迭代次数 4: 得分 = 0.670818425474
迭代次数 5: 得分 = 0.667348864361
平均得分 0.672467413212
测试集 AUC 是: 0.6797
```

1. blending

下面介绍模型融合之 blending 方法。本例中，第 1 层模型选择的是支持向量机、朴素贝叶斯、K 最近邻、随机森林，第 2 层模型选择的是逻辑回归。

第 1 层：

```
from sklearn.svm import SVC                                          #支持向量机
from sklearn.naive_bayes import GaussianNB as GNB                    #朴素贝叶斯
from sklearn.neighbors import KNeighborsClassifier as KNN            #K 最近邻
from sklearn.ensemble import RandomForestClassifier as RFC           #随机森林
from sklearn.linear_model import LogisticRegression as LR            #逻辑回归
svc = SVC(probability=True)
gnb = GNB()
knn = KNN()
rfc = RFC(random_state=1)
model = [svc, gnb, knn, rfc]
X_train_d1, X_train_d2, y_train_d1, y_train_d2 = sp(X_train, y_train, test_size=0.5,
           random_state=1)
X_train_d2_blending = np.zeros((X_train_d2.shape[0], len(model)))
X_test_blending = np.zeros((X_test.shape[0], len(model)))
for j, clf in enumerate(model):
    clf.fit(X_train_d1, y_train_d1)
    y_test_value = clf.predict_proba(X_train_d2)[:, 1]
    #模型列表中每个模型的预测结果,组成新的训练集,用于第 2 层模型
    X_train_d2_blending[:, j] = y_test_value
    #模型列表中每个模型的预测结果,组成新的测试集,用于第 2 层模型
    X_test_blending[:, j] = clf.predict_proba(X_test)[:, 1]
    print('测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, X_test_blending[:,
j])))
```

输出结果如下：

```
测试集 AUC 是：0.6301
测试集 AUC 是：0.6521
测试集 AUC 是：0.6105
测试集 AUC 是：0.7953
```

第 2 层：

```
from sklearn.linear_model import LogisticRegression as LR
lr = LR()
lr.fit(X_train_d2_blending, y_train_d2)
y_test_value = lr.predict_proba(X_test_blending)[:, 1]
y_test_value = (y_test_value - y_test_value.min()) / (y_test_value.max() - y_test_value.min())
print('融合后,测试集 AUC 是：{:.4}'.format(metrics.roc_auc_score(y_test, y_test_value)))
```

输出结果如下：

```
融合后,测试集 AUC 是：0.7943
```

由此可见，默认参数的逻辑回归算出测试集上的 AUC 是 0.6815，融合模型算出来的 AUC 是 0.7943，此次模型融合预测结果提升了 16.55 个百分点。

2. stacking

下面介绍 stacking 方法，本例中的模型列表与 blending 方法一样，采用的是支持向量机、朴素贝叶斯、*K* 最近邻和随机森林。

第 1 层：

```
X_train = np.array(X_train)
y_train = np.array(y_train)
X_train_stacking = np.zeros((X_train.shape[0], len(model)))
X_test_stacking = np.zeros((X_test.shape[0], len(model)))
from sklearn.cross_validation import StratifiedKFold as SKF
skf = list(SKF(y_train, 5))
for j, clf in enumerate(model):
    X_test_stacking_j = np.zeros((X_test.shape[0], len(skf)))
    for i, (fold1, fold2) in enumerate(skf):
        X_train_fold, y_train_fold, X_test_fold, y_test_fold = X_train[fold1], y_train[fold1],
                     X_train[fold2], y_train[fold2]
        clf.fit(X_train_fold, y_train_fold)
        y_test_value = clf.predict_proba(X_test_fold)[:, 1]
        X_train_stacking[fold2, j] = y_test_value
```

```
        X_test_stacking_j[:, i] = clf.predict_proba(X_test)[:, 1]
    X_test_stacking[:, j] = X_test_stacking_j.mean(axis = 1)
    print('测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, X_test_stacking[:,j])))
```

预测结果如下：

```
测试集 AUC 是: 0.6578
测试集 AUC 是: 0.6635
测试集 AUC 是: 0.6346
测试集 AUC 是: 0.8557
```

第 2 层：

```
from sklearn.linear_model import LogisticRegression as LR
lr = LR()
lr.fit(X_train_stacking, y_train)
y_test_value = lr.predict_proba(X_test_stacking)[:, 1]
#  标准化
y_test_value = (y_test_value - y_test_value.min()) / (y_test_value.max() - y_test_value.min())
print('融合后,测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, y_test_value)))
```

预测结果如下：

```
融合后,测试集 AUC 是: 0.8561
```

此次逻辑回归结果相对于 blending 又有了较大的提升。

关于 stacking 方法，有个封装好的第三方包 mlxtend，首先安装 mlxtend，安装代码为：pip install mlxtend，然后导入包，进行分析，代码如下：

```
from mlxtend.classifier import StackingClassifier
from sklearn.model_selection import cross_val_score
sclf = StackingClassifier(classifiers = model, meta_classifier = lr, use_probas = True)
for clf, label in zip([svc, gnb, knn, rfc, sclf], ['svc', 'gnb', 'knn', 'rfc', 'sclf']):
    scores = cross_val_score(clf, X_train, y_train, cv = 5, scoring = 'roc_auc')
    print('roc_auc {}, {}'.format(scores.mean(), label))
```

运行结果如下：

```
roc_auc 0.6413919395505555, svc
roc_auc 0.6606336495667565, gnb
roc_auc 0.6203925329163198, knn
roc_auc 0.8019715220818838, rfc
roc_auc 0.8028352114075513, sclf
```

不难发现,模型效果还是不错的,融合后的 sclf 模型指标高于其他模型。

6.2 回归模型的融合方法



视频讲解

上面介绍了分类模型的模型融合方法,现在介绍一下回归模型的融合方法。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import os
os.chdir('../data')
data = pd.read_csv('data2.csv')
allFeatures = list(data.columns)
allFeatures.remove('target')
allFeatures.remove('id')
X = data[allFeatures]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size=0.3, random_state=1)
'''定义评价指标'''
from sklearn.model_selection import cross_val_score
def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y_train, scoring=
                "neg_mean_squared_error", cv = 5))
    return rmse
```

这里的评价指标是交叉验证后的均方误差。

下面开始建模,先看看单个模型的验证效果,代码如下:

```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Lasso                  #对异常值敏感
from sklearn.preprocessing import RobustScaler          #取四分位差
lasso = make_pipeline(RobustScaler(), Lasso(random_state=1))
score = rmse_cv(lasso)
print('lasso 分数是{:.2f}'.format(score.mean()))
from sklearn.ensemble import RandomForestRegressor as RFR
rfr = RFR()
score = rmse_cv(rfr)
print('rfr 分数是{:.2f}'.format(score.mean()))
from sklearn.ensemble import GradientBoostingRegressor as GBR
```

```
gbr = GBR()
score = rmse_cv(gbr)
print('gbr 分数是{:.2f}'.format(score.mean()))
import xgboost as xgb
xgbr = xgb.XGBRegressor()
score = rmse_cv(xgbr)
print('分数是{:.2f}'.format(score.mean()))
import lightgbm as lgb
lgbr = lgb.LGBMRegressor()
score = rmse_cv(lgbr)
print('分数是{:.2f}'.format(score.mean()))
```

运行结果是：

```
lasso 分数是 2725.84
rfr 分数是 2489.27
gbr 分数是 2502.07
xgb 分数是 2502.23
lgb 分数是 2321.25
```

下面开始使用 Stacking 方法进行模型融合。

Stacking1：这里写了一个类，原理是多个基学习器对数据进行训练和预测，然后根据学习器的预测结果，求出平均值，将其作为最终的预测结果。

```
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models
    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]
        for model in self.models_:
            model.fit(X, y)
        return self
    def predict(self, X):
        #纵向合并 1 维数组
        predictions = np.column_stack([model.predict(X) for model in self.models_])
        return np.mean(predictions, axis=1)
averaged_models = AveragingModels(models = (lasso, gbr, xgbr, lgbr))
score = rmse_cv(averaged_models)
print('分数是{:.2f}'.format(score.mean()))
```

运行结果是：

```
分数是 2468.91
```

下面看第二个 Stacking 方法，使用 5 折交叉验证，各模型预测结果取平均值，并由元模型预测。

```
from sklearn.model_selection import KFold
class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds
    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, random_state=1)
        out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
        for i, model in enumerate(self.base_models):
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(model)
                self.base_models_[i].append(instance)
                instance.fit(X[train_index], y[train_index])
                y_pred = instance.predict(X[holdout_index])
                out_of_fold_predictions[holdout_index, i] = y_pred
        self.meta_model_.fit(out_of_fold_predictions, y)
        return self
    def predict(self, X):
        meta_features = np.column_stack([np.column_stack([model.predict(X) for model
                in base_models]).mean(axis=1) for base_models in self.base_models_ ])
        return self.meta_model_.predict(meta_features)
stacked_averaged_models = StackingAveragedModels(base_models = (gbr, xgbr, lgbr),
                                                  meta_model = lasso)
score = rmse_cv(averaged_models)
print('分数是{:.2f}'.format(score.mean()))
```

预测结果是：

```
分数是 2468.76
```

6.3 小结

以上的 Stacking 方法是可以直接使用的，具有普适性。以上代码也许不是最好的，在特定场景下也许不是最优的。那么，做到在已有代码的基础上，理解、提炼、创新出更好的代码与算法，才是优秀风控建模师能力的体现。

第 7 章

创建金融申请评分卡



视频讲解

业内某知名人士曾说过这么一段话，“以信用评分模型的方式来进行风险管理，这是国际上比较成功的实践经验。由于消费信贷业务具有笔数多、单笔金额小、数据维度丰富的特征，决定了需要对其实行智能化、概率化的管理模式。信用评分模型运用现代的数理统计技术，通过对消费者信用历史记录和业务活动记录的深度挖掘、分析和提炼，从而发现蕴藏在纷繁复杂数据中的消费者风险特征，并通过评分的方式总结出来，进而作为管理决策的科学依据。这种管理方法，已成为国际上普遍运用的风险管理最佳操作典范。除了风险管理这一核心领域，这种智能化、概率化的管理模式还被贯穿到消费信贷和信用卡的收益管理、客户忠诚度管理、客户关系管理等方面，涵盖了信贷生命周期管理的各个阶段，包括从产品设计到市场营销、从信贷审批到账户管理、从坏账催收到反欺诈等。在这些方面，国际上都有比较成功的经验值得我们借鉴。”

说到金融领域的评分卡，就不得不提到美国的 FICO。自美国三大征信局形成体系之后，FICO 总结了征信数据的所有重要规律，以一个简单扼要的评分方式对 2 亿多美国人进行一个准确的风险评估。这就是如今一个普通美国人能够快速地获得普

惠金融服务的最根本条件。如果用户要去银行申请贷款只需要银行查一下用户的信用评分是多少，就会知道你是否符合审批标准，能批多少额度，而且利率也会非常贴近用户真实的风险水平。这是美国金融市场的一个重大里程碑。

如今，FICO是全球最大的个人信用评分机构，每年向市场提供上百亿个FICO评分，管理着全球65%的信用卡，其采用的很多消费信贷和信用卡业务的技术手段都成为了行业标准。

评分卡按照应用场景分类，可以分风险评分卡、营销响应评分卡、客户流失评分卡、员工离职倾向评分卡等；按照用途和时间的不同，风险评分卡又可以分为贷前的申请评分卡、申请欺诈评分卡，贷中的行为评分卡以及贷后的催收评分卡。

本章主要介绍风险评分卡里的申请评分卡，这是最常用也是最重要的评分卡之一。

首先必须要明白两个概念：观察期和表现期。

观察期主要用于收集信用历史和行为特征等信息，从而提炼出能够预测未来信用表现的预测变量。一般来说，根据产品的不同，可以将放款前的6～12个月作为观察期，用来获取特征变量。

表现期主要用于收集信用表现的信息，如是否拖欠、是否逾期、是否流失等，从而提炼表现变量，确定目标客户。

FICO中国区总裁陈建先生曾经说过，申请风险评分模型的表现变量的定义一般根据表现期终的信用表现界定如下：

(1) 3期拖欠以上、呆账、破产的账户定义为“坏”。

(2) 未拖欠或1期拖欠的账户定义为“好”。

(3) 2期拖欠的账户定义为“不确定”，被排除于模型之外。

(4) 如果银行业务历史较短，“坏”的样本量不足(一般一个模型需要800～1500个“坏”账户为样本)，则可以把2期拖欠的账户也定义为“坏”。

(5) 如果按“表现期终”的界定方法“坏”样本量不足，也可以把“表现期内”最大拖欠达3期以上的统统定义为“坏”(即使其在表现期终恢复到相对“好”的地位)。

以上这段话更多地倾向于长期循环贷或银行信用卡产品。从实际应用层面上来说，根据各信贷产品的应用场景不同，目标变量是不能一概而论的。目标客户的定义可以根据业务规则强制来定，也可以根据坏客户迁徙矩阵来定；表现期的长短除了根据自身的历史数据定义以外，还可以把逾期率趋于稳定的时间窗口定义为表现期。

再说说观察期与特征变量。申请评分卡时，各特征必须是贷前就能获取的特征，一般围绕着客户的还款能力、还款意愿、稳定性和多头借贷这4点展开。选择数据供

应商时，除了上述 4 个核心点，还需要兼顾自己的产品客群定位、用户体验、数据成本。比如产品是农户贷，那么社保、公积金、网银工资流水就是不需要获取的特征。如果产品目标是电商小商户，则电商数据就是非常有用的特征。

那么，明确表现期和观察期的概念以后，该如何创建申请评分卡呢？

创建申请评分卡有两种方法：一种是逻辑回归模型；另一种是机器学习模型。

逻辑回归模型是个非常好的模型，效果非常稳定，也容易从业务角度进行解释，只要通过简单的变换，就能得到入模各变量的评分，但是它对数据的质量要求较高，还要服从前提假设，并且精度不是很高。

再说说机器学习模型，申请评分本质上是违约概率的映射，违约概率完全可以用机器学习模型计算出来，而且准确度也较高。可是机器学习模型未能完全代替逻辑回归模型，其主要原因是模型很难从业务角度进行解释，并且模型部署难度较高。因此，目前申请评分卡主要还是使用逻辑回归模型。那么本章就详细介绍一下如何用 Python 创建逻辑回归模型下的金融申请评分卡。

逻辑回归模型创建金融申请评分卡的核心步骤，如下所述。

(1) 变量分箱。常用的分箱方法是等宽、等频、人工指定分箱切割点、卡方分箱、C4.5 决策树分箱这 5 种。前面 3 种方法是无监督分箱法，需要人工主动调试，更多地依赖于建模人员的经验和对业务的理解；后两种分箱法又称为最优分箱法。

(2) WOE 编码(Weight of Evidence)，又称为证据权重。计算公式是每箱好样本比例与坏样本比例的比值的自然对数。

(3) IV 值(Information Value)，又称为信息浓度。计算公式是每箱好样本比例与坏样本比例的差值，再乘以对应的 WOE 值。一般选择 IV≥0.02 的变量。

(4) 共线性、相关性、显著性检验。

(5) 计算每个变量对应切分点的分数。

7.1 变量选择

在构建逻辑回归模型时，从观察期收集的指标可能有几百个，那么就需要从收集到的所有指标中筛选出对违约状态影响最大的指标，作为入模指标，来开发模型，这就是变量选择。

```
##############################################################
#   Step 1: LightGBM 重要性选择变量
##############################################################
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import os
os.chdir('../data')
data = pd.read_csv('data_all_values.csv')
```

预览数据，如图 7-1 所示。

```
In [3]: data.shape
Out[3]: (103434, 19)

In [4]: data.head()
Out[4]:
```

	贷款金额	贷款期限	利率	每月还款金额	贷款等级	工作年限	房屋所有权	年收入	收入是否由LC验证	target	贷款目的	月负债比	过去两年借款人逾期30天以上的数字	过去6个月内被查询次数	摧毁公共记录的数量	额度循环使用率	总贷款笔数	拖欠的逾期款项	留置税数量
0	18000.0	1	7.99	563.98	1	2.0	2	60000.0	3	0	debt_consolidation	30.76	0.0	0	0.0	17.8	22.0	0.0	0.0
1	13000.0	1	5.32	391.50	1	10.0	2	101800.0	3	0	credit_card	18.91	0.0	0	3.0	57.5	21.0	0.0	0.0
2	5000.0	1	10.49	162.49	2	10.0	3	75000.0	3	0	other	11.01	0.0	1	1.0	47.3	9.0	0.0	0.0
3	11100.0	1	5.32	334.28	1	3.0	3	60000.0	3	0	credit_card	18.91	0.0	0	0.0	34.4	23.0	0.0	0.0
4	25000.0	2	28.69	788.84	6	10.0	3	83807.4	1	0	debt_consolidation	13.17	0.0	0	0.0	38.6	25.0	0.0	0.0

图 7-1　评分卡数据预览展示图

这是个有 103434 行、19 列的数据。

目标变量的水平分布如下：

```
data['target'].value_counts()
0      95087
1      8347
Name: target, dtype: int64
```

先把变量分成连续变量与分类变量，分类依据是连续变量的水平种类在 10 种以上，分类变量的水平种类在 10 种以下。

```
data_copy = data.copy()
allFeatures = list(data.columns)
allFeatures.remove('target')
for i in allFeatures:
    print('变量: {}的不同水平值有{}个'.format(i, len(data[i].unique())))
```

运行结果如下：

```
变量：贷款金额的不同水平值有 1481 个
变量：贷款期限的不同水平值有 2 个
变量：利率的不同水平值有 57 个
变量：每月还款金额的不同水平值有 15228 个
变量：贷款等级的不同水平值有 7 个
变量：工作年限的不同水平值有 10 个
变量：房屋所有权的不同水平值有 3 个
变量：年收入的不同水平值有 8454 个
变量：收入是否由 LC 验证的不同水平值有 3 个
变量：贷款目的的不同水平值有 13 个
变量：月负债比的不同水平值有 4196 个
变量：过去两年借款人逾期 30 天以上的数字的不同水平值有 22 个
变量：过去 6 个月内被查询次数的不同水平值有 6 个
变量：摧毁公共记录的数量的不同水平值有 19 个
变量：额度循环使用率的不同水平值有 1085 个
变量：总贷款笔数的不同水平值有 108 个
变量：拖欠的逾期款项的不同水平值有 420 个
变量：留置税数量的不同水平值有 18 个
```

所以，分类变量和连续变量分别如下：

```
categorical_var = ['贷款期限', '贷款等级', '工作年限', '房屋所有权', '收入是否由 LC 验证','贷款目的', '过去 6 个月内被查询次数', '留置税数量']
continuous_var = ['贷款金额', '利率', '每月还款金额', '年收入', '月负债比', '过去两年借款人逾期 30 天以上的数字', '摧毁公共记录的数量', '额度循环使用率', '总贷款笔数', '拖欠的逾期款项']
```

首先进行变量筛选。这里使用 LightGBM 算法先进行重要性排序，再对变量进行粗筛。

```
'''连续变量标准化'''
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data[continuous_var] = sc.fit_transform(data[continuous_var])
'''数值分类变量转整型'''
string_var = list(data.select_dtypes(include = ["object"]).columns)
col = list(set(categorical_var) - set(string_var))
data[col] = data[col].astype(int)
'''字符分类变量按照坏样本率进行编码'''
def Encoder(df, col, target):
    encoder = {}
    for v in set(df[col]):
        if v == v:
            subDf = df[df[col] == v]
        else:
```

```
            xList = list(df[col])
            nanInd = [i for i in range(len(xList)) if xList[i] != xList[i]]
            subDf = df.loc[nanInd]
        encoder[v] = sum(subDf[target]) * 1.0/subDf.shape[0]
        newCol = [encoder[i] for i in df[col]]
        return newCol
string_var = list(data.select_dtypes(include = ["object"]).columns)
col = list(set(categorical_var) & set(string_var))
for i in col:
      data[i] = Encoder(data, i, 'target')
```

现在再预览一下数据，如图 7-2 所示。

```
In [9]: data.head()
Out[9]:
```

	贷款金额	贷款期限	利率	每月还款金额	贷款等级	工作年限	房屋所有权	年收入	收入是否由LC验证	target	贷款目的	月负债比	过去两年借款人逾期30天以上的数字	过去6个月内被查询次数	摧毁公共记录的数量	额度循环使用率	总贷款笔数	拖欠的逾期款项	留置税数量
0	0.417400	1	-1.142543	0.465334	1	2	2	-0.268366	3	0	0.084803	1.032461	-0.381038	0	-0.401308	-1.349822	-0.166258	-0.022457	0
1	-0.125116	1	-1.680914	-0.151171	1	10	2	0.272899	3	0	0.066410	0.012113	-0.381038	0	4.053237	0.295178	-0.250177	-0.022457	0
2	-0.993140	1	-0.638450	-0.969736	2	10	3	-0.074132	3	0	0.080146	-0.668120	-0.381038	1	1.083540	-0.127467	-1.257209	-0.022457	0
3	-0.331272	1	-1.680914	-0.355696	1	3	3	-0.268366	3	0	0.066410	0.012113	-0.381038	0	-0.401308	-0.661988	-0.082339	-0.022457	0
4	1.176921	2	3.031349	1.269065	6	10	3	0.039914	1	0	0.084803	-0.482132	-0.381038	0	-0.401308	-0.487958	0.085500	-0.022457	0

图 7-2 评分卡数据标准化之后的预览展示图

LightGBM 模型有个非常好的功能，就是能指定分类变量，省去了 one-hot 编码的步骤。

```
'''指定整型分类变量作为 LightGBM 的分类特征'''
col = list(set(categorical_var) - set(string_var))
'''保存变量和文件'''
import pickle
f = open('lgb_col.pkl','wb')
pickle.dump(col,f)
f.close()
'''建模'''
allFeatures = list(data.columns)
allFeatures.remove('target')
X = data[allFeatures]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size = 0.3, random_state = 1)
'''加载分类变量'''
f = open('lgb_col.pkl','rb')
col = pickle.load(f)
f.close()
```

```
'''LightGBM 建模'''
import lightgbm as LGB
params = {
          'objective': 'binary',
          'boosting' : 'gbdt',
          'num_leaves': 4,
          'min_data_in_leaf': 20,
          'subsample': 0.9,
          'colsample_bytree': 0.8,
          'learning_rate':0.09,
          'tree_learner': 'voting',
          'metric': 'auc'
          }
dtrain = LGB.Dataset(X_train, y_train, categorical_feature = col)
dtest = LGB.Dataset(X_test, y_test, reference = dtrain, categorical_feature = col)
lgb = LGB.train(params, dtrain, valid_sets = [dtrain, dtest], num_boost_round = 3000,
                early_stopping_rounds = 100, verbose_eval = 10)
```

运行结果如下：

```
Training until validation scores don't improve for 100 rounds.
[10] training's auc: 0.670416 valid_1's auc: 0.668585
[20] training's auc: 0.67935 valid_1's auc: 0.675068
[30] training's auc: 0.68141 valid_1's auc: 0.67786
[40] training's auc: 0.684492 valid_1's auc: 0.679672
[50] training's auc: 0.687648 valid_1's auc: 0.681506
[60] training's auc: 0.689867 valid_1's auc: 0.682638
[70] training's auc: 0.691886 valid_1's auc: 0.683422
[80] training's auc: 0.693716 valid_1's auc: 0.683969
[90] training's auc: 0.695131 valid_1's auc: 0.684526
[100] training's auc: 0.69629 valid_1's auc: 0.68495
[110] training's auc: 0.697477 valid_1's auc: 0.685131
[120] training's auc: 0.69868 valid_1's auc: 0.685219
[130] training's auc: 0.699542 valid_1's auc: 0.685314
[140] training's auc: 0.700352 valid_1's auc: 0.685386
[150] training's auc: 0.701254 valid_1's auc: 0.685586
[160] training's auc: 0.702065 valid_1's auc: 0.685466
[170] training's auc: 0.702856 valid_1's auc: 0.685529
[180] training's auc: 0.70363 valid_1's auc: 0.685604
[190] training's auc: 0.704406 valid_1's auc: 0.685665
[200] training's auc: 0.704822 valid_1's auc: 0.68562
[210] training's auc: 0.705378 valid_1's auc: 0.685495
[220] training's auc: 0.70624 valid_1's auc: 0.685289
[230] training's auc: 0.706735 valid_1's auc: 0.685263
[240] training's auc: 0.707306 valid_1's auc: 0.685158
[250] training's auc: 0.7078 valid_1's auc: 0.68513
[260] training's auc: 0.708471 valid_1's auc: 0.685116
```

```
[270] training's auc: 0.709135 valid_1's auc: 0.684981
[280] training's auc: 0.709565 valid_1's auc: 0.685044
[290] training's auc: 0.710201 valid_1's auc: 0.684811
Early stopping, best iteration is:
[191] training's auc: 0.704434 valid_1's auc: 0.685696
```

说明最好的迭代轮次是第 191 轮，最好的结果是训练集 auc 是 0.70，最好的测试集 auc 是 0.68。

下面计算变量重要性。

```
'''LightGBM 重要性选择变量'''
importace = list(lgb.feature_importance())
allFeatures = list(lgb.feature_name())
featureImportance = zip(allFeatures, importace)
featureImportanceSorted = sorted(featureImportance, key = lambda k: k[1], reverse = True)
plt.figure(figsize = (5, 10))
sns.barplot(x = [k[1] for k in featureImportanceSorted], y = [k[0] for k in
            featureImportanceSorted])
plt.xticks(rotation = 'vertical')
plt.show()
```

运行结果如图 7-3 所示。



图 7-3　LightGBM 重要性排序展示图

这里选择前 13 个最重要的变量。

```
feature_selection_lgb = [k[0] for k in featureImportanceSorted[:13]]
```

7.2 各变量按照 ln(odds)进行分箱

odds 是好坏比之意，其计算公式为：odds＝好客户比例/坏客户比例。如果该箱好客户多，则 ln(odds)＞0；反之，ln(odds)＜0。

```
##########################################################
#   Step 2: 分类变量分箱
##########################################################
data = data_copy.copy()
data = data[feature_selection_lgb + ['target']]
data_copy = data.copy()
'''分类变量按照 ln(odds)进行编码'''
def Ln_odds(df, col, target):
    total = df.groupby([col])[target].count()
    total = pd.DataFrame({'total': total})
    bad = df.groupby([col])[target].sum()
    bad = pd.DataFrame({'bad': bad})
    regroup = total.merge(bad, left_index = True, right_index = True, how = 'left')
    regroup.reset_index(level = 0, inplace = True)
    N = sum(regroup['total'])
    B = sum(regroup['bad'])
    regroup['good'] = regroup['total'] - regroup['bad']
    G = N - B
    regroup['bad_pcnt'] = regroup['bad'].map(lambda x: x * 1.0/B)
    regroup['good_pcnt'] = regroup['good'].map(lambda x: x * 1.0 / G)
    regroup[col + '_WOE'] = regroup.apply(lambda x: np.log(x.good_pcnt * 1.0/x.bad_pcnt),
                                  axis = 1)
    df = pd.merge(df, regroup[[col,col + '_WOE']], on = col, how = 'left')
    return df
'''分类变量分箱'''
categorical_var = list(set(categorical_var) & set(feature_selection_lgb))
categorical_var = ['贷款期限', '房屋所有权', '贷款等级', '过去 6 个月内被查询次数', '工作
年限']
for i in categorical_var:
      data = Ln_odds(data, i, 'target')
      sns.pointplot(x = i, y = i + '_WOE', data = data)
      plt.xticks(rotation = 0)
      plt.show()
```

运行结果如图 7-4 所示。

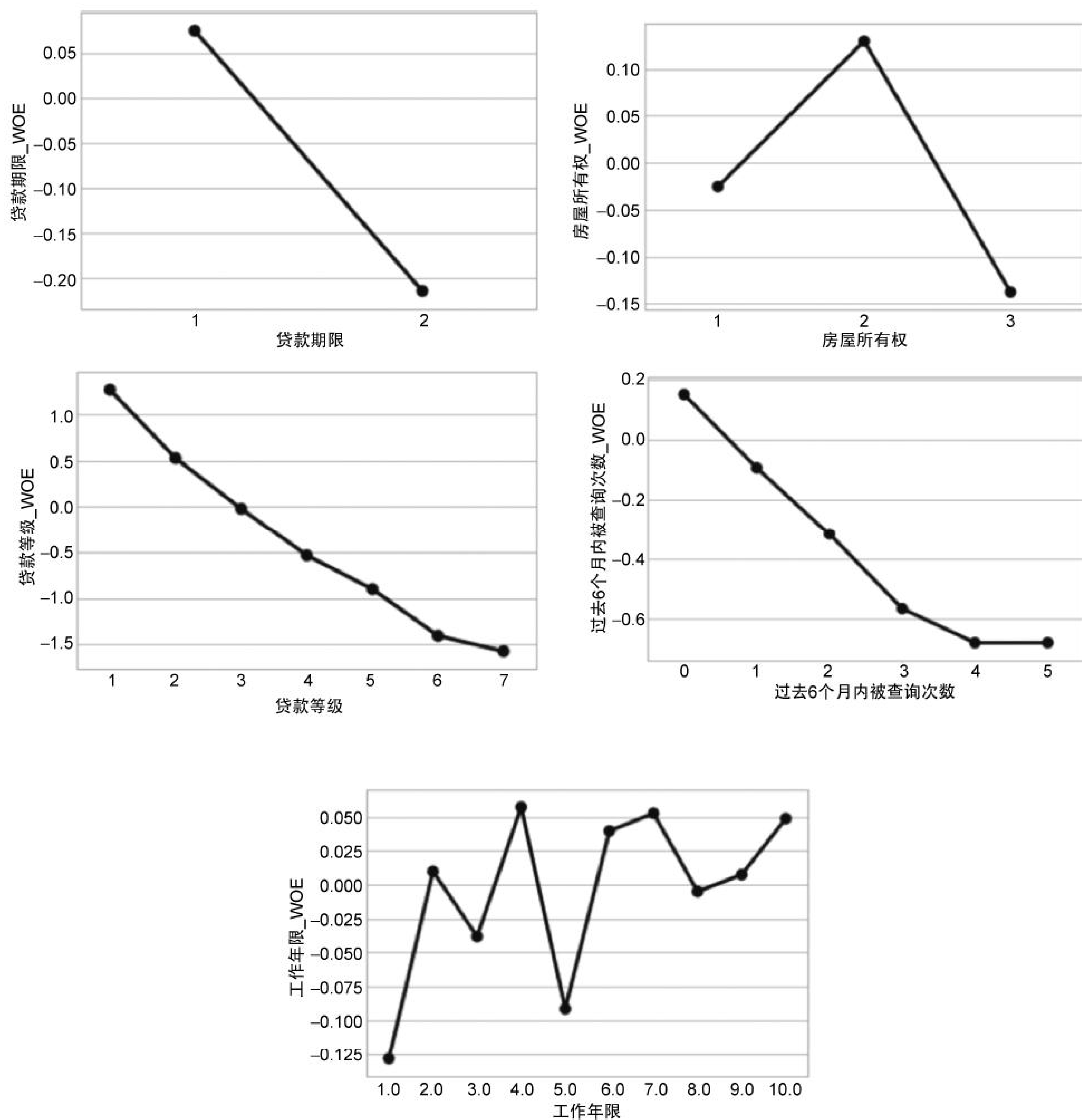


图 7-4　分类变量的 WOE 编码分箱展示图

在变量进行分箱时，一般最多分成 5 箱，且分箱结果必须单调，否则将进行水平合并或重新分箱。下面逐个进行调试。

1. 变量“贷款等级”分箱

```
'''逐个调试'''
data = data_copy.copy()
i = '贷款等级'
data[i] = data[i].apply(lambda x: 2 if 2<=x<5 else x)
```

```
data[i] = data[i].apply(lambda x: 3 if x>=5 else x)
data = Ln_odds(data, i, 'target')
sns.pointplot(x=i, y=i+'_WOE', data=data)
plt.xticks(rotation=0)
plt.show()
```

变量“贷款等级”分箱结果如图 7-5 所示。

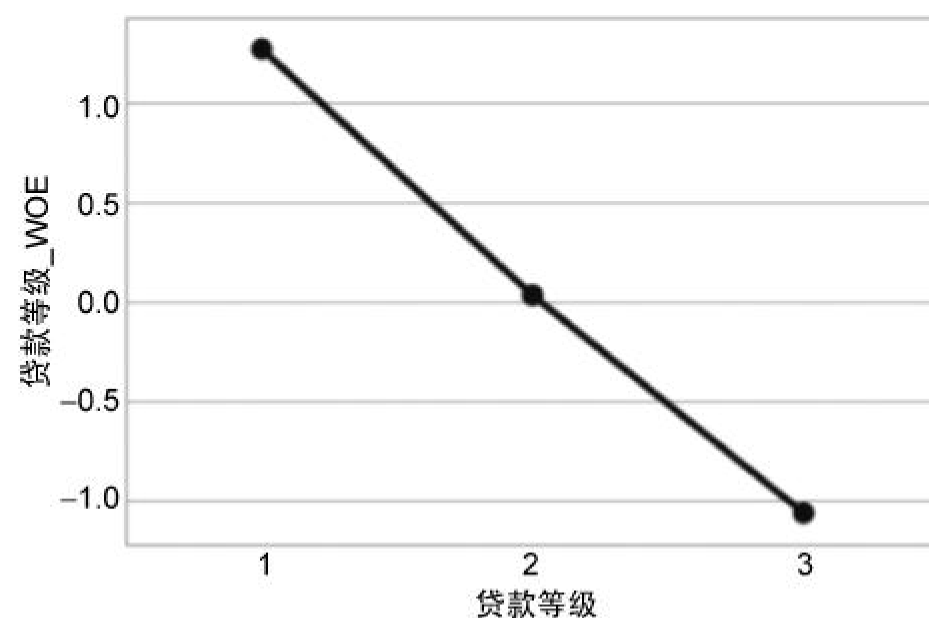


图 7-5 “贷款等级”的 WOE 编码分箱展示图

2. 变量“过去 6 个月内被查询次数”分箱

```
i = '过去6个月内被查询次数'
data[i] = data[i].apply(lambda x: 4 if x>3 else x)
data = Ln_odds(data, i, 'target')
sns.pointplot(x=i, y=i+'_WOE', data=data)
plt.xticks(rotation=0)
plt.show()
```

变量“过去 6 个月内被查询次数”分箱结果如图 7-6 所示。

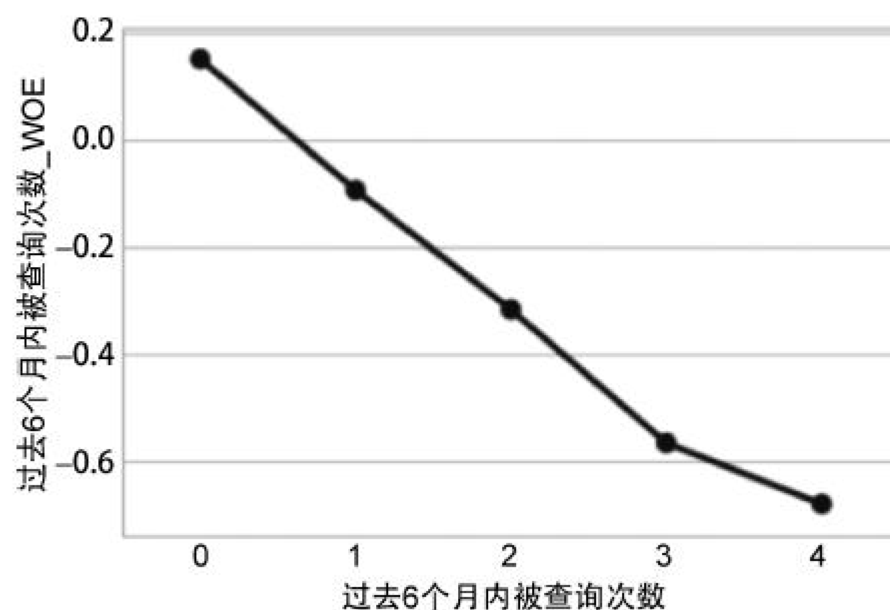


图 7-6 “过去 6 个月内被查询次数”的 WOE 编码分箱展示图

3. 变量“工作年限”分箱

```
i = '工作年限'
data[i] = data[i].apply(lambda x: 1 if 1<=x<5 else x)
data[i] = data[i].apply(lambda x: 2 if x>=5 else x)
data = Ln_odds(data, i, 'target')
sns.pointplot(x=i, y=i+'_WOE', data=data)
plt.xticks(rotation=0)
plt.show()
```

变量“工作年限”分箱结果如图 7-7 所示。

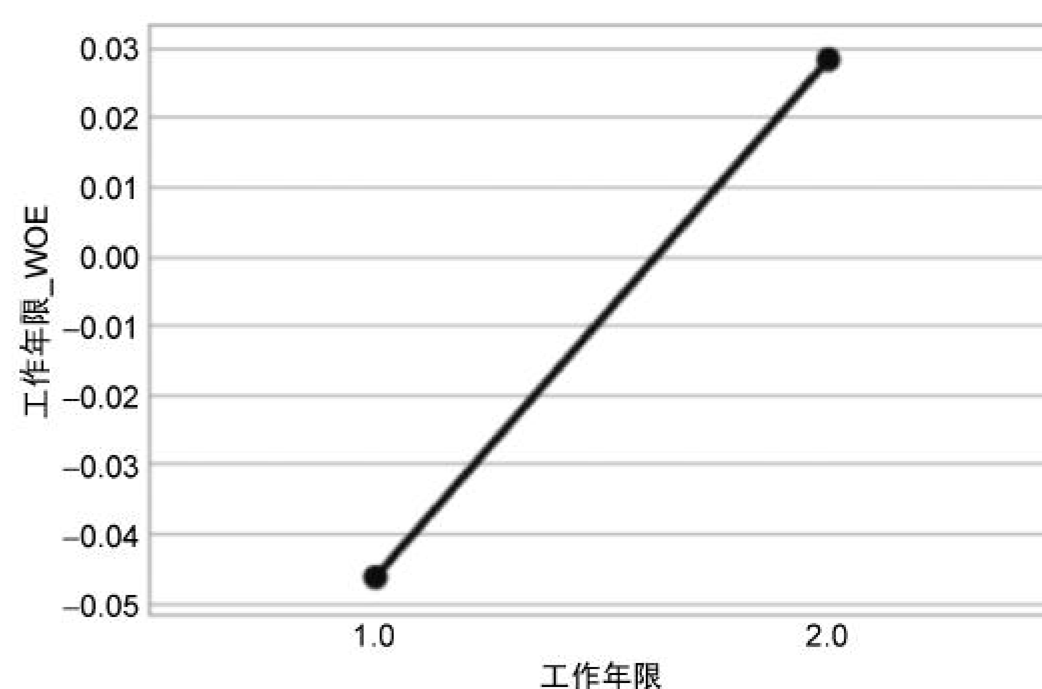


图 7-7　“工作年限”的 WOE 编码分箱展示图

4. 连续变量分箱

(1) 变量“总贷款笔数”分箱

```
######################################################
#  Step 3: 连续变量分箱
######################################################
data = pd.read_csv('分类变量分箱.csv')
continuous_var = list(set(continuous_var) & set(feature_selection_lgb))
continuous_var = ['总贷款笔数', '每月还款金额', '过去两年借款人逾期 30 天以上的数字',
                  '贷款金额', '年收入', '利率', '月负债比', '额度循环使用率']
describe = data[continuous_var].describe().T[['max','min']]
'''连续变量分箱'''
i = '总贷款笔数'
sns.distplot(data[i][data['target'] == 0].dropna(),color='blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color='red')
plt.show()
bins = [-1, 20, 30, 200] #左开右闭,如果可能出现 0,那么需要以 -1 开始
```

```
cats = pd.cut(list(data[i]), bins, precision = 0)
cats.value_counts()
data[i + '组别'] = pd.Series(cats)
data = Ln_odds(data, i + '组别', 'target')
sns.pointplot(x = i + '组别', y = i + '组别_WOE', data = data.sort_values(i + '组别_WOE',
             ascending = False))
plt.show()
```

变量“总贷款笔数”分箱结果如图 7-8 所示。

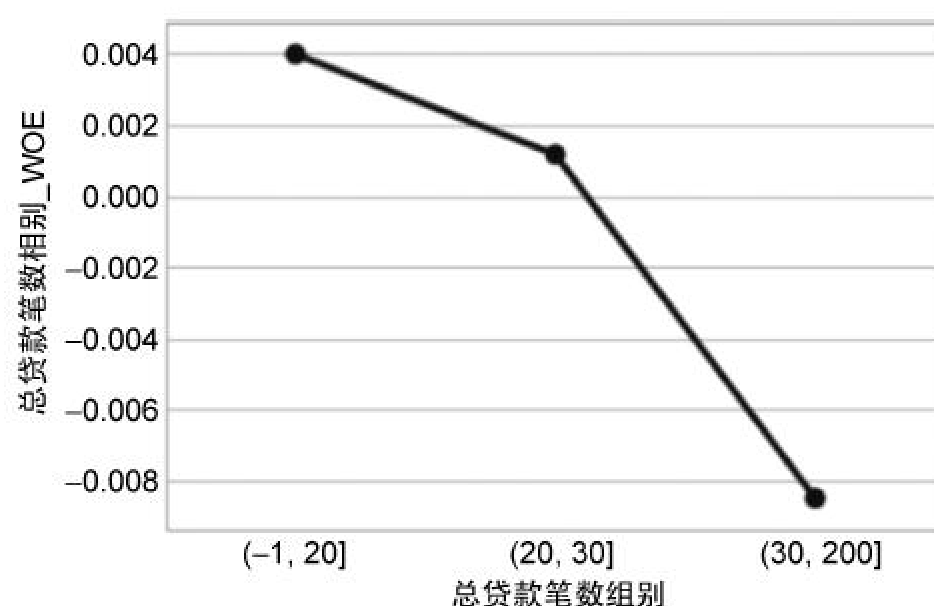


图 7-8 “总贷款笔数”组别的 WOE 编码分箱展示图

（2）变量“每月还款金额”分箱

```
i = '每月还款金额'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.show()
bins = [-1, 300, 750, 2000]
cats = pd.cut(list(data[i]), bins, precision = 0)
cats.value_counts()
data[i + '组别'] = pd.Series(cats)
data = Ln_odds(data, i + '组别', 'target')
sns.pointplot(x = i + '组别', y = i + '组别_WOE', data = data.sort_values(i + '组别_WOE',
             ascending = False))
plt.show()
```

变量“每月还款金额”分箱结果如图 7-9 所示。

（3）变量“过去两年借款人逾期 30 天以上的数字”分箱

```
i = '过去两年借款人逾期 30 天以上的数字'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.show()
bins = [-1, 1, 50]
```

```
cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()
data[i+'组别'] = pd.Series(cats)
data = Ln_odds(data, i+'组别', 'target')
sns.pointplot(x=i+'组别', y=i+'组别_WOE', data=data.sort_values(i+'组别_WOE',
              ascending=False))
plt.show()
```

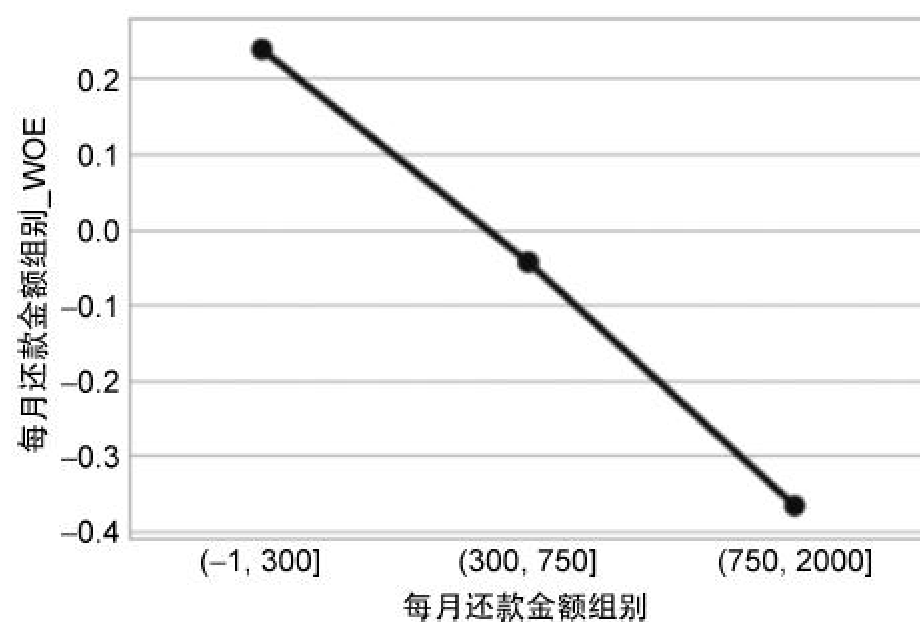


图 7-9 “每月还款金额”组别的 WOE 编码分箱展示图

变量“过去两年借款人逾期 30 天以上的数字”分箱结果如图 7-10 所示。

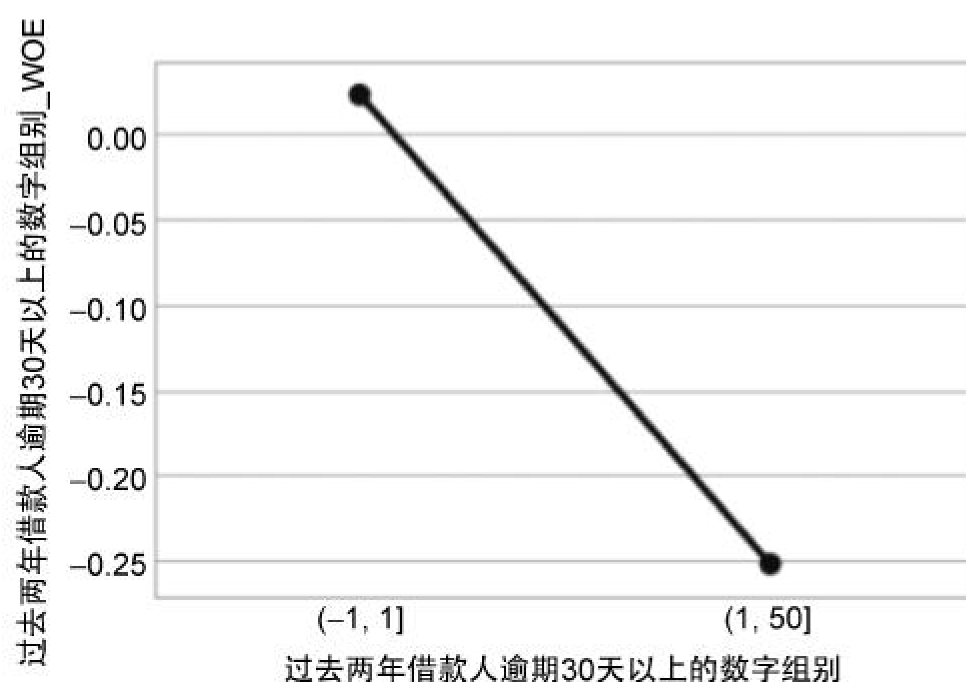


图 7-10 “过去两年借款人逾期 30 天以上的数字”组别的 WOE 编码分箱展示图

(4) 变量“贷款金额”分箱

```
i = '贷款金额'
sns.distplot(data[i][data['target'] == 0].dropna(),color='blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color='red')
plt.show()
bins = [-1, 10000, 20000, 50000]
cats = pd.cut(list(data[i]), bins, precision=0)
cats.value_counts()
data[i+'组别'] = pd.Series(cats)
```

```
data = Ln_odds(data, i + '组别', 'target')
sns.pointplot(x = i + '组别', y = i + '组别_WOE', data = data.sort_values(i + '组别_WOE',
              ascending = False))
plt.show()
```

变量“贷款金额”分箱结果如图 7-11 所示。

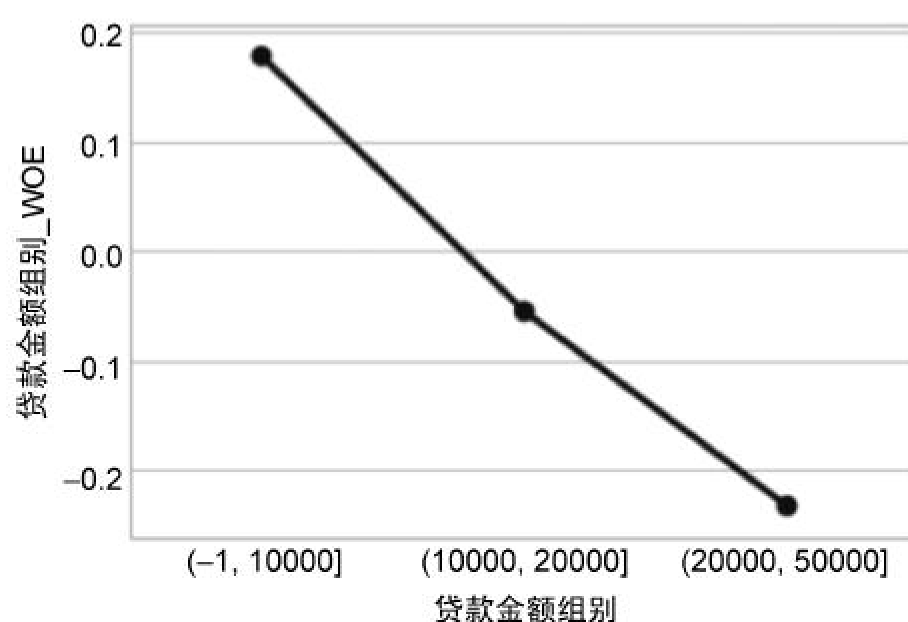


图 7-11 “贷款金额”组别的 WOE 编码分箱展示图

(5) 变量“年收入”分箱

```
i = '年收入'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.xlim([0,1000000])
plt.show()
bins = [-1, 150000, 300000, 10000000]
cats = pd.cut(list(data[i]), bins, precision = 0)
cats.value_counts()
data[i + '组别'] = pd.Series(cats)
data = Ln_odds(data, i + '组别', 'target')
sns.pointplot(x = i + '组别', y = i + '组别_WOE', data = data.sort_values(i + '组别_WOE',
              ascending = False))
plt.show()
```

变量“年收入”分箱结果如图 7-12 所示。

(6) 变量“利率”分箱

```
i = '利率'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.show()
bins = [0, 8, 13, 50]
cats = pd.cut(list(data[i]), bins, precision = 0)
cats.value_counts()
```

```
data[i + '组别'] = pd.Series(cats)
data = Ln_odds(data, i + '组别', 'target')
sns.pointplot(x = i + '组别', y = i + '组别_WOE', data = data.sort_values(i + '组别_WOE',
            ascending = False))
plt.show()
```

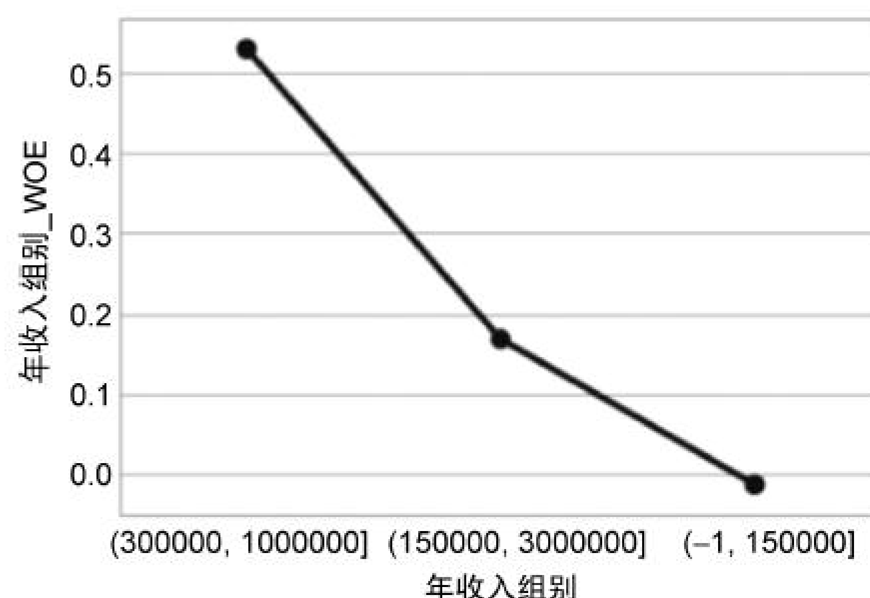


图 7-12　“年收入”组别的 WOE 编码分箱展示图

变量“利率”分箱结果如图 7-13 所示。

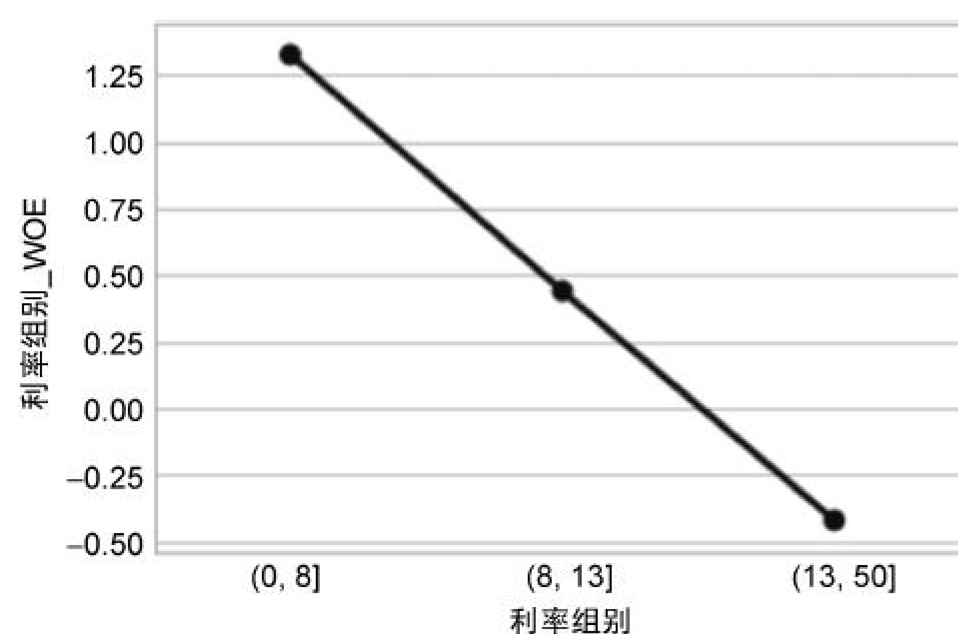


图 7-13　“利率”组别的 WOE 编码分箱展示图

（7）变量“月负债比”分箱

```
i = '月负债比'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.xlim([0,200])
plt.show()
bins = [ - 1, 20, 1000]
cats = pd.cut(list(data[i]), bins, precision = 0)
cats.value_counts()
data[i + '组别'] = pd.Series(cats)
data = Ln_odds(data, i + '组别', 'target')
sns.pointplot(x = i + '组别', y = i + '组别_WOE', data = data.sort_values(i + '组别_WOE',
```

```
                ascending = False))
plt.show()
```

变量“月负债比”分箱结果如图 7-14 所示。

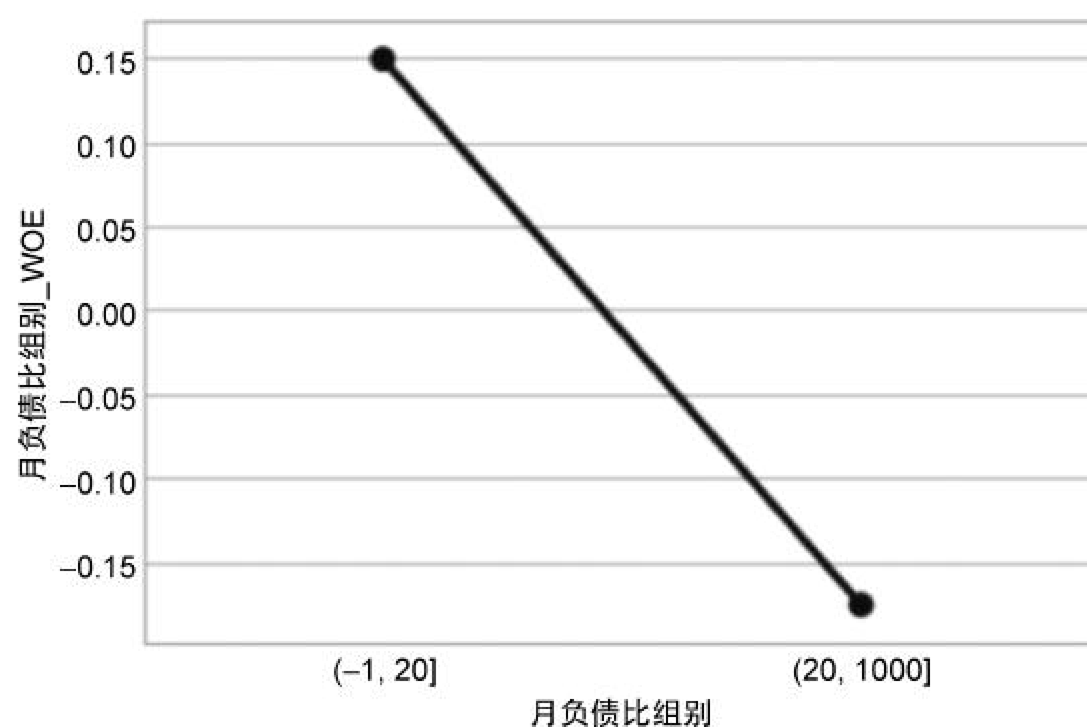


图 7-14 “月负债比”组别的 WOE 编码分箱展示图

(8) 变量“额度循环使用率”分箱

```
i = '额度循环使用率'
sns.distplot(data[i][data['target'] == 0].dropna(),color = 'blue')
sns.distplot(data[i][data['target'] == 1].dropna(),color = 'red')
plt.show()
bins = [-1, 50, 200]
cats = pd.cut(list(data[i]), bins, precision = 0)
cats.value_counts()
data[i + '组别'] = pd.Series(cats)
data = Ln_odds(data, i + '组别', 'target')
sns.pointplot(x = i + '组别', y = i + '组别_WOE', data = data.sort_values(i + '组别_WOE',
                ascending = False))
plt.show()
```

变量“额度循环使用率”分箱结果如图 7-15 所示。

```
'''保存文件'''
col = []
for i in list(data.columns):
      if i.find('_WOE')< 0:
              col.append(i)
data = data[col]
data.to_csv('分箱完成.csv', index = False, encoding = 'utf-8')
```

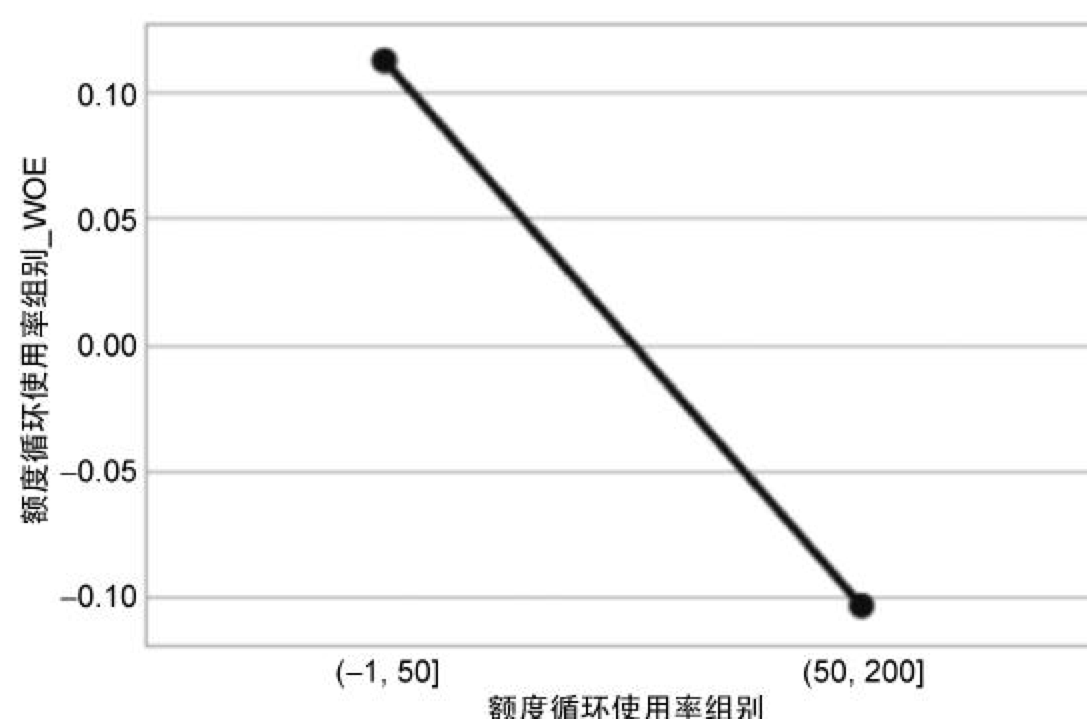


图 7-15　“额度循环使用率”组别的 WOE 编码分箱展示图

7.3　计算 WOE 与 IV 值

证据权重（weight of evidence，WOE）用于计算每箱的好坏比，WOE 的数值越大，表明每箱中的好样本越多。IV 值就是 WOE 值对应的信息浓度。

```
#######################################################
#   Step 4: 计算 WOE 和 IV
#######################################################
def CalcWOE(df, col, target):
    total = df.groupby([col])[target].count()
    total = pd.DataFrame({'total': total})
    bad = df.groupby([col])[target].sum()
    bad = pd.DataFrame({'bad': bad})
    regroup = total.merge(bad, left_index = True, right_index = True, how = 'left')
    regroup.reset_index(level = 0, inplace = True)
    N = sum(regroup['total'])          #计算总样本数
    B = sum(regroup['bad'])            #计算坏样本数
    regroup['good'] = regroup['total'] - regroup['bad']
    G = N - B
    regroup['bad_pcnt'] = regroup['bad'].map(lambda x: x * 1.0/B)
    regroup['good_pcnt'] = regroup['good'].map(lambda x: x * 1.0 / G)
    regroup['WOE'] = regroup.apply(lambda x: np.log(x.good_pcnt * 1.0/x.bad_pcnt),axis = 1)
    WOE_dict = regroup[[col, 'WOE']].set_index(col).to_dict(orient = 'index')
    for k, v in WOE_dict.items():
        WOE_dict[k] = v['WOE']
        IV = regroup.apply(lambda x: (x.good_pcnt - x.bad_pcnt) *
                           np.log(x.good_pcnt * 1.0/x.bad_pcnt), axis = 1)
        IV = sum(IV)
    return {"WOE": WOE_dict, 'IV': IV}
```

```
all_var = []
for i in list(data.columns):
    if i.find('组别') > 0:
        all_var.append(i)
all_var = all_var + categorical_var
WOE_dict = {}
IV_dict = {}
for var in all_var:
    woe_iv = CalcWOE(data, var, 'target')
    WOE_dict[var] = woe_iv['WOE']
    IV_dict[var] = woe_iv['IV']
```

现在预览一下数据,如图 7-16 所示。

```
In [36]: data.head()
Out[36]:
```

	利率	每月还款金额	额度循环使用率	月负债比	总贷款笔数	贷款金额	年收入	贷款期限	贷款等级	过去两年借款人逾期30天以上的数字	...	工作年限	target	总贷款笔数组别	每月还款金额组别	过去两年借款人逾期30天以上的数字组别	贷款金额组别	年收入组别	利率组别	月负债比组别	额度循环使用率组别
0	7.99	563.98	17.8	30.76	22.0	18000.0	60000.0	1	1	0.0	...	1.0	0	(20, 30]	(300, 750]	(-1, 1]	(10000, 20000]	(-1, 150000]	(0, 8]	(20, 1000]	(-1, 50]
1	5.32	391.50	57.5	18.91	21.0	13000.0	101800.0	1	1	0.0	...	2.0	0	(20, 30]	(300, 750]	(-1, 1]	(10000, 20000]	(-1, 150000]	(0, 8]	(-1, 20]	(50, 200]
2	10.49	162.49	47.3	11.01	9.0	5000.0	75000.0	1	2	0.0	...	2.0	0	(-1, 20]	(-1, 300]	(-1, 1]	(-1, 10000]	(-1, 150000]	(8, 13]	(-1, 20]	(-1, 50]
3	5.32	334.28	34.4	18.91	23.0	11100.0	60000.0	1	1	0.0	...	1.0	0	(20, 30]	(300, 750]	(-1, 1]	(10000, 20000]	(-1, 150000]	(0, 8]	(-1, 20]	(-1, 50]
4	28.69	788.84	38.6	13.17	25.0	25000.0	83807.4	2	3	0.0	...	2.0	0	(20, 30]	(750, 2000]	(-1, 1]	(20000, 50000]	(-1, 150000]	(13, 50]	(-1, 20]	(-1, 50]

5 rows × 22 columns

图 7-16 评分卡数据预览展示图

7.4 逻辑回归建模

逻辑回归(Logistic Regression),虽然名字上带有“回归”的字样,但其实质却是一种常用的分类模型,主要用于二分类问题。逻辑回归可以通过 logistics 函数,将特征空间映射成某些事件发生的概率。计算出变量 WOE 值和 IV 值之后,就可以对 WOE 变量进行 logistics 映射,并将其转换为逾期概率。

```
############################################################
#   Step 5: 逻辑回归建模
############################################################
'''选取 IV>=0.02 的变量'''
IV_dict_sorted = sorted(IV_dict.items(), key=lambda x: x[1], reverse=True)
IV_name = [i[0] for i in IV_dict_sorted]
IV_values = [i[1] for i in IV_dict_sorted]
```

```
high_IV = {k:v for k, v in IV_dict.items() if v >= 0.02}
high_IV_sorted = sorted(high_IV.items(), key = lambda x:x[1], reverse = True)
print ('总共有',len(high_IV_sorted),'个变量的 IV >= 0.02')
```

运行结果如下：

```
总共有 6 个变量的 IV >= 0.02
'''创建 WOE 变量'''
short_list = high_IV.keys()
short_list_2 = []
for var in short_list:
    newVar = var + '_WOE'
    data[newVar] = data[var].map(WOE_dict[var])
    short_list_2.append(newVar)
'''计算相关系数矩阵,删除与重要业务变量相关性较强的自变量'''
dataWOE = data[short_list_2]
corr = round(dataWOE.corr(),2)
mask = np.zeros_like(corr, dtype = np.bool)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize = (5, 5))
cmap = sns.diverging_palette(220, 10, as_cmap = True)
sns.heatmap(corr,mask = mask, cmap = cmap, center = 0, annot = True, cbar_kws = {"shrink": .5})
plt.show()
```

运行结果如图 7-17 所示。

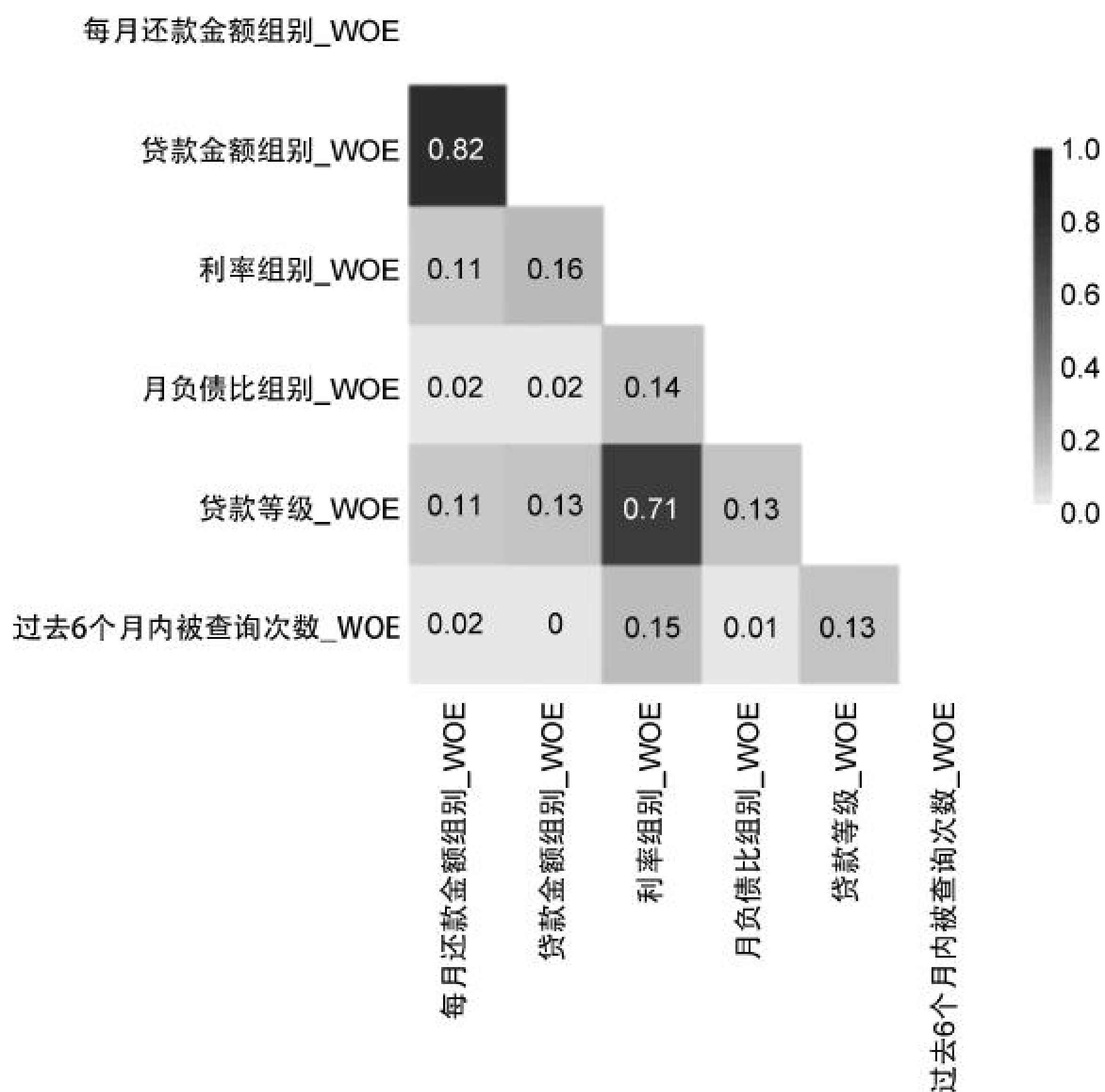


图 7-17　6 个变量的 WOE 相关系数图

计算相关系数之后，还需要考虑多重共线性问题。评价多重共线性的指标主要是方差膨胀因子。方差膨胀因子（Variance Inflation Factor，VIF）是指解释变量之间存在多重共线性时的方差与不存在多重共线性时的方差之比。VIF 值越大，代表共线性越严重。经验判断方法表明：当 0＜VIF＜10，不存在多重共线性；当 10≤VIF＜100，存在较强的多重共线性；当 VIF≥100，存在严重多重共线性。

```
"""选择方差共线性< 10 的变量"""
col = np.array(data[short_list_2])
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
for i in range(len(short_list_2)):
    print ('{} VIF 是{}'.format(short_list_2[i], vif(col, i)))
```

运行结果如下：

```
每月还款金额组别_WOE VIF 是 3.1504765201473144
贷款金额组别_WOE VIF 是 3.1868414919401897
利率组别_WOE VIF 是 2.1775374815158592
月负债比组别_WOE VIF 是 1.0255937275175637
贷款等级_WOE VIF 是 2.1297714474712253
过去 6 个月内被查询次数_WOE VIF 是 1.0298508719860746
```

在确定变量之间不存在多重共线性之后，就要对变量进行显著性分析，删除 P 值不显著的变量。

```
"""判断显著性"""
X = data[short_list_2]
X['intercept'] = [1] * X.shape[0]
y = data['target']
import statsmodels.api as sm
lr_sm = sm.Logit(y, X).fit()
lr_sm.summary()
```

运行结果如图 7-18 所示。

最后建模，代码如下：

```
X = data[short_list_2]
y = data['target']
from sklearn.cross_validation import train_test_split as sp
X_train, X_test, y_train, y_test = sp(X, y, test_size = 0.3, random_state = 1)
from sklearn.linear_model import LogisticRegression as LR
```

Out[51]:

Logit Regression Results

Dep. Variable:	target	No. Observations:	103434
Model:	Logit	Df Residuals:	103427
Method:	MLE	Df Model:	6
Date:	Sun, 18 Mar 2018	Pseudo R-squ.:	0.04845
Time:	16:33:07	Log-Likelihood:	-27605.
converged:	True	LL-Null:	-29010.
		LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
每月还款金额组别_WOE	-1.0481	0.098	-10.661	0.000	-1.241	-0.855
贷款金额组别_WOE	0.6825	0.124	5.497	0.000	0.439	0.926
利率组别_WOE	-0.6757	0.029	-23.313	0.000	-0.733	-0.619
月负债比组别_WOE	-0.5186	0.072	-7.213	0.000	-0.660	-0.378
贷款等级_WOE	-0.5519	0.029	-18.903	0.000	-0.609	-0.495
过去6个月内被查询次数_WOE	-0.6137	0.053	-11.500	0.000	-0.718	-0.509
intercept	-2.4398	0.012	-202.610	0.000	-2.463	-2.416

图 7-18 6个变量的显著性结果展示图

```
lr = LR(random_state = 1)
lr.fit(X_train, y_train)
from sklearn import metrics
y_test_label = lr.predict(X_test)
y_test_value = lr.predict_proba(X_test)[:, 1]
print('测试集准确率是: {:.2%}'.format(metrics.accuracy_score(y_test, y_test_label)))
print('测试集 AUC 是: {:.4}'.format(metrics.roc_auc_score(y_test, y_test_value)))
```

运行结果如下：

```
测试集准确率是: 91.82%
测试集 AUC 是: 0.6683
```

7.5 创建评分卡

在确定模型的准确度和 AUC 的可接受范围后，可以对每个入模变量进行单变量得分计算，从而求出变量不同取值下的得分。

```
#############################################################
#   Step 6: 创建评分卡
#############################################################
b = lr.intercept_                              #截距
```

```
coe = lr.coef_                              # 系数
a0 = coe[0][0]                              # 第一个变量的系数
a1 = coe[0][1]
a2 = coe[0][2]
a3 = coe[0][3]
a4 = coe[0][4]
a5 = coe[0][5]
A = 500
PDO = 20                                    # 每增加 20 分,odds 增加 1 倍
B = PDO/np.log(2)
'''创建 每月还款金额 单变量得分'''
WOE_dict['每月还款金额组别']                  # 获取字典 key,即变量水平值
woe1 = WOE_dict['每月还款金额组别']['(-1, 300]']
score1 = -(B * a0 * woe1) + (A-B*b)/dataWOE.shape[0]
woe2 = WOE_dict['每月还款金额组别']['(300, 750]']
score2 = -(B * a0 * woe2) + (A-B*b)/dataWOE.shape[0]
woe3 = WOE_dict['每月还款金额组别']['(750, 2000]']
score3 = -(B * a0 * woe3) + (A-B*b)/dataWOE.shape[0]
''' 对应的 SQL 语句
case
    when 0 <= '每月还款金额'<= 300 then 7
    when 300 < '每月还款金额'<= 750 then -1
    when '每月还款金额'> 750 then -11
else 0                                      # 以业务逻辑或补缺规则来定
'''
'''创建 贷款金额 单变量得分'''
WOE_dict['贷款金额组别']
woe1 = WOE_dict['贷款金额组别']['(-1, 10000]']
score1 = -(B * a1 * woe1) + (A-B*b)/dataWOE.shape[0]
woe2 = WOE_dict['贷款金额组别']['(10000, 20000]']
score2 = -(B * a1 * woe2) + (A-B*b)/dataWOE.shape[0]
woe3 = WOE_dict['贷款金额组别']['(20000, 50000]']
score3 = -(B * a1 * woe3) + (A-B*b)/dataWOE.shape[0]
''' 对应的 SQL 语句
case
     when 0 <= '贷款金额'<= 10000 then -3
     when 10000 < '贷款金额'<= 20000 then 1
     when '贷款金额'> 20000 then 4
else 0
'''
'''创建 利率 单变量得分'''
WOE_dict['利率组别']
woe1 = WOE_dict['利率组别']['(0, 8]']
score1 = -(B * a2 * woe1) + (A-B*b)/dataWOE.shape[0]
woe2 = WOE_dict['利率组别']['(8, 13]']
score2 = -(B * a2 * woe2) + (A-B*b)/dataWOE.shape[0]
woe3 = WOE_dict['利率组别']['(13, 50]']
score3 = -(B * a2 * woe3) + (A-B*b)/dataWOE.shape[0]
''' 对应的 SQL 语句
case
    when 0 < '利率'<= 8 then 25
```

```
    when 8 < '利率'<= 13 then 8
    when '利率'> 13 then -8
else 0
'''
'''创建 月负债比 单变量得分'''
WOE_dict['月负债比组别']
woe1 = WOE_dict['月负债比组别']['(-1, 20]']
score1 = -(B * a3 * woe1) + (A-B*b)/dataWOE.shape[0]
woe2 = WOE_dict['月负债比组别']['(20, 1000]']
score2 = -(B * a3 * woe2) + (A-B*b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
    when 0 < '月负债比'<= 20 then 2
    when '月负债比'> 20 then -3
else 0
'''
'''创建 贷款等级 单变量得分'''
WOE_dict['贷款等级']
woe1 = WOE_dict['贷款等级'][1]
score1 = -(B * a4 * woe1) + (A-B*b)/dataWOE.shape[0]
woe2 = WOE_dict['贷款等级'][2]
score2 = -(B * a4 * woe2) + (A-B*b)/dataWOE.shape[0]
woe3 = WOE_dict['贷款等级'][3]
score3 = -(B * a4 * woe3) + (A-B*b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
    when '贷款等级' = 1 then 20
    when '贷款等级' = 2 then 0
    when '贷款等级' = 3 then -17
else 0
'''
'''创建 过去6个月内被查询次数 单变量得分'''
WOE_dict['过去6个月内被查询次数']
woe1 = WOE_dict['过去6个月内被查询次数'][0]
score1 = -(B * a5 * woe1) + (A-B*b)/dataWOE.shape[0]
woe2 = WOE_dict['过去6个月内被查询次数'][1]
score2 = -(B * a5 * woe2) + (A-B*b)/dataWOE.shape[0]
woe3 = WOE_dict['过去6个月内被查询次数'][2]
score3 = -(B * a5 * woe3) + (A-B*b)/dataWOE.shape[0]
woe4 = WOE_dict['过去6个月内被查询次数'][3]
score4 = -(B * a5 * woe4) + (A-B*b)/dataWOE.shape[0]
woe5 = WOE_dict['过去6个月内被查询次数'][4]
score5 = -(B * a5 * woe5) + (A-B*b)/dataWOE.shape[0]
'''对应的 SQL 语句
case
     when '过去6个月内被查询次数' = 1 then 3
     when '过去6个月内被查询次数' = 2 then -2
     when '过去6个月内被查询次数' = 3 then -5
     when '过去6个月内被查询次数' = 4 then -10
     when '过去6个月内被查询次数' = 5 then -12
else 0
'''
```

如此，可根据 case when 算出每个区间的得分，然后将得分相加，即可得到总分，该总分即为金融申请评分卡的最终分数。

```
SELECT LOAN_NO,(每月还款金额 + 贷款金额 + 利率 + 月负债比 + 贷款等级 + 过去 6 个月内被查询
次数) AS risk_score
FROM
(SELECT
    LOAN_NO,
    CASE
       WHEN 0 <=  每月还款金额 <=  300 THEN 7
       WHEN 300 < 每月还款金额 <=  750 THEN  - 1
       WHEN 每月还款金额 > 750 THEN  - 11
       ELSE 0                              #以业务逻辑或补缺规则来定
    END 每月还款金额,
    CASE
       WHEN 0 <=  贷款金额 <=  10000 THEN  - 3
       WHEN 10000 < 贷款金额 <=  20000 THEN 1
       WHEN 贷款金额 > 20000 THEN 4
       ELSE 0
    END 贷款金额,
    CASE
       WHEN 0 < 利率 <=  8 THEN 25
       WHEN 8 < 利率 <=  13 THEN 8
       WHEN 利率 > 13 THEN  - 8
       ELSE 0
    END 利率,
    CASE
       WHEN 0 <月负债比<=  20 THEN 2
       WHEN 月负债比> 20 THEN  - 3
       ELSE 0
    END 月负债比,
    CASE
       WHEN 贷款等级 =  1 THEN 20
       WHEN 贷款等级 =  2 THEN 0
       WHEN 贷款等级 = 3 THEN  - 17
       ELSE 0
    END 贷款等级,
    CASE
       WHEN 过去 6 个月内被查询次数  =  1 THEN 3
       WHEN 过去 6 个月内被查询次数  =  2 THEN  - 2
       WHEN 过去 6 个月内被查询次数  =  3 THEN  - 5
       WHEN 过去 6 个月内被查询次数  =  4 THEN  - 10
       WHEN 过去 6 个月内被查询次数  =  5 THEN  - 12
       ELSE 0
    END 过去 6 个月内被查询次数,
FROM 数据库中对应的表) GROUP BY LOAN_NO
```

7.6 申请评分卡的评价、使用与监控

评分卡创建成功后,该如何评价、使用和监控呢?

评价方法除了二分类模型通用的 auc、ks 指标以外,还有区分度曲线、拟合度曲线。使用的主要方法是风险评分分布表。

评分卡监控主要从两方面来衡量,一个是稳定性指标,一个是有效性指标。

模型的稳定性指标主要是指 PSI 值,一般来说,如果 PSI 值在 0.25 以下,则说明模型是稳定的。除此以外,还可以做出变量稳定性分析报告和评分分布报告。

模型的有效性指标主要是指 kendalls'tau 值,值越接近 1,说明逾期率在分数上的单调下降性越明显,分数越有效。除此以外,还可以做出模型拟合度曲线、逾期拖欠分布报告与模型区分度检验报告。

7.7 小结

本章主要介绍了申请评分卡创建的整个过程,从好坏样本的定义到最终的各变量得分都进行了详细讲解。在建模过程中,一定要细心地处理数据,搞清楚业务逻辑,这样得出的评分对于业务才具有实际意义。

第 8 章

社交网络分析与反欺诈



视频讲解

在实际应用中，创建社交网络需要专门的图数据库，其中 Neo4j 就是最常用的图数据库之一。Neo4j 最大的好处就是可以快速地、直观地匹配节点与节点之间的关系。Neo4j 节点间的关系图如图 8-1 所示，中间的节点连接了很多其他的节点。

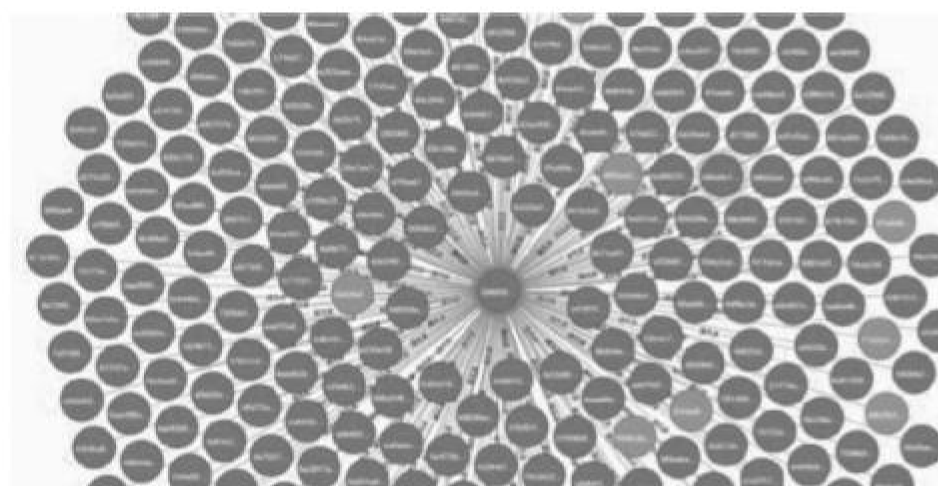


图 8-1　Neo4j 节点间的关系图

在金融领域，可以利用社交网络查找失联用户、发现欺诈社区、找到活跃中介。在其他领域，可以用 Neo4j 构造实时推荐引擎，创建六度人脉社交网络，创建工商企业控制人图谱等。本书介绍的 Neo4j，一方面是各大公司越来越看重的社交网络图谱，另一方面是可以使用 Python 与 Neo4j 数据库进行连接，能够很方便地与数据库进行交互，只需要下载一个第三方包 Py2neo 即可，安装语句为：pip install py2neo。

8.1　Neo4j 的下载与安装

Neo4j 有很多版本，有企业服务器版，有社区版，还有个人桌面版。一般来说，如果仅是个人单机使用，使用个人桌面版是最方便的；如果需要运维人员部署到服务器中，则需要下载安装社区版；如果数据量特别大，那么可能就要付费购买专用企业服务器版了。

Neo4j 的官方链接地址为：https://neo4j.com/，如图 8-2 所示，单击右上角的 Download 按钮，即可进入下载页面，如图 8-3 所示，单击左下角的 DOWNLOAD NEO4J SERVER 按钮，可以看到更多的 Neo4j 版本，如图 8-4 所示。安装步骤详见附录 D。

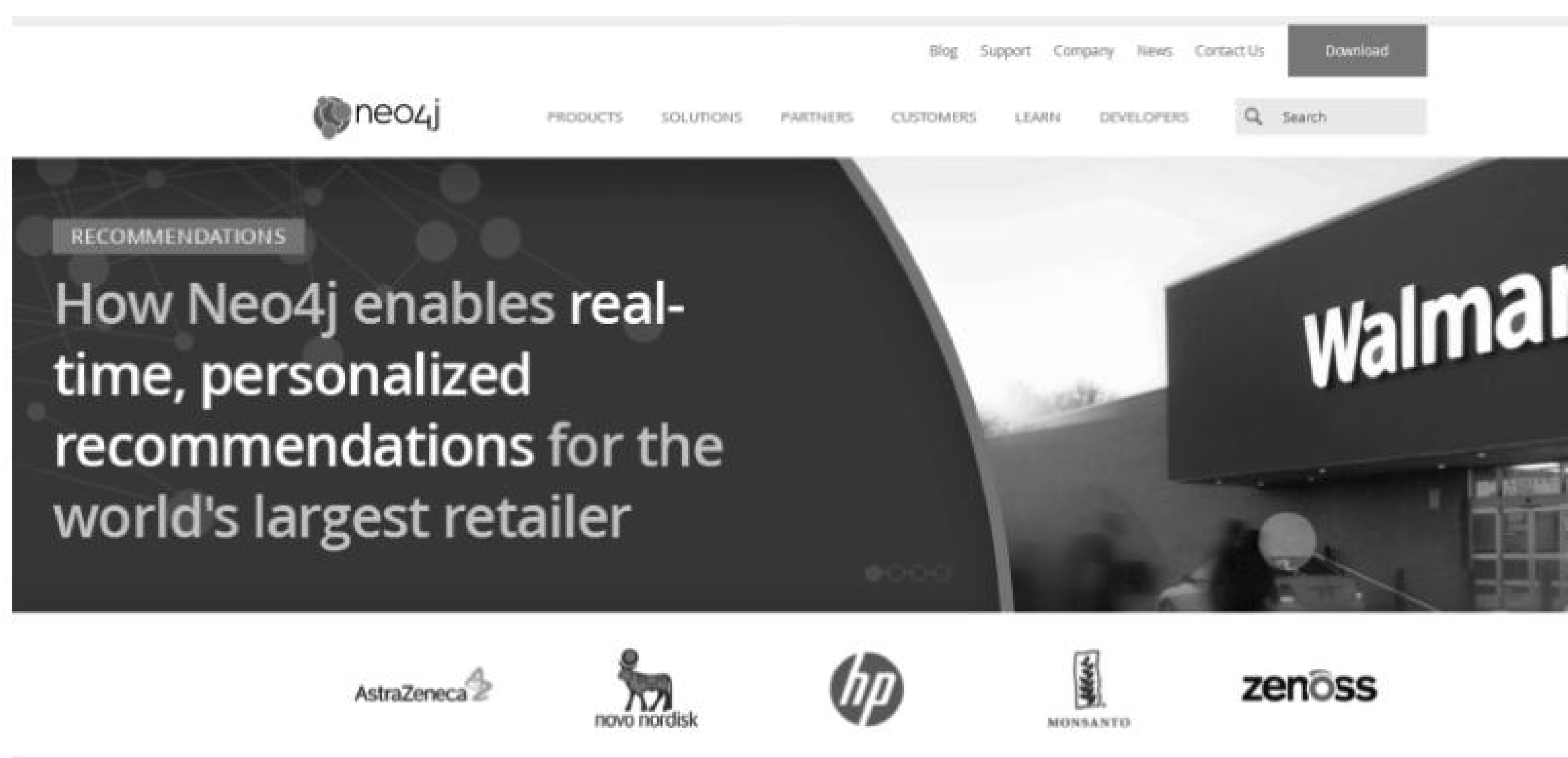


图 8-2　Neo4j 官方首页展示图

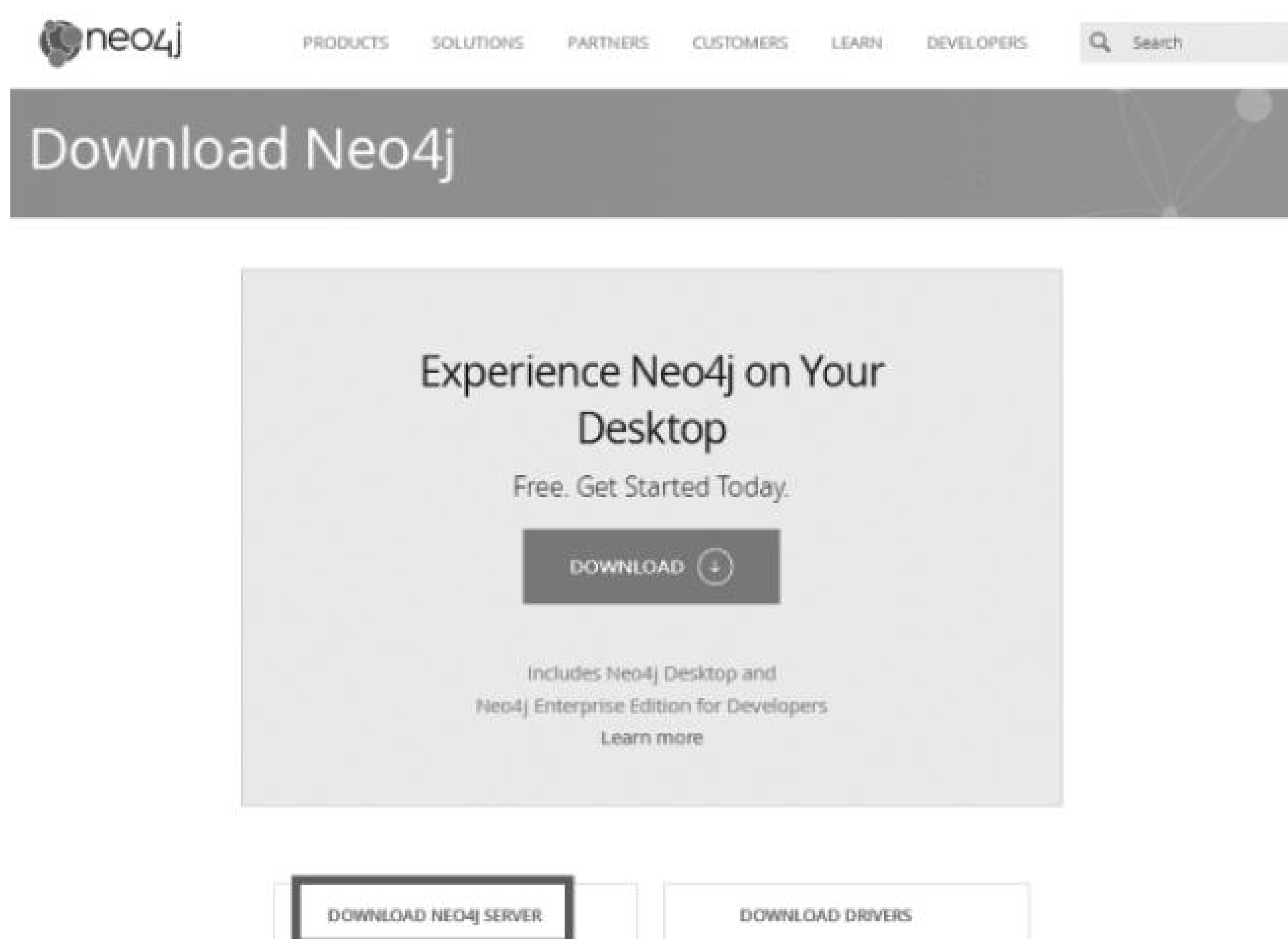


图 8-3　Neo4j 官方下载链接页面展示图

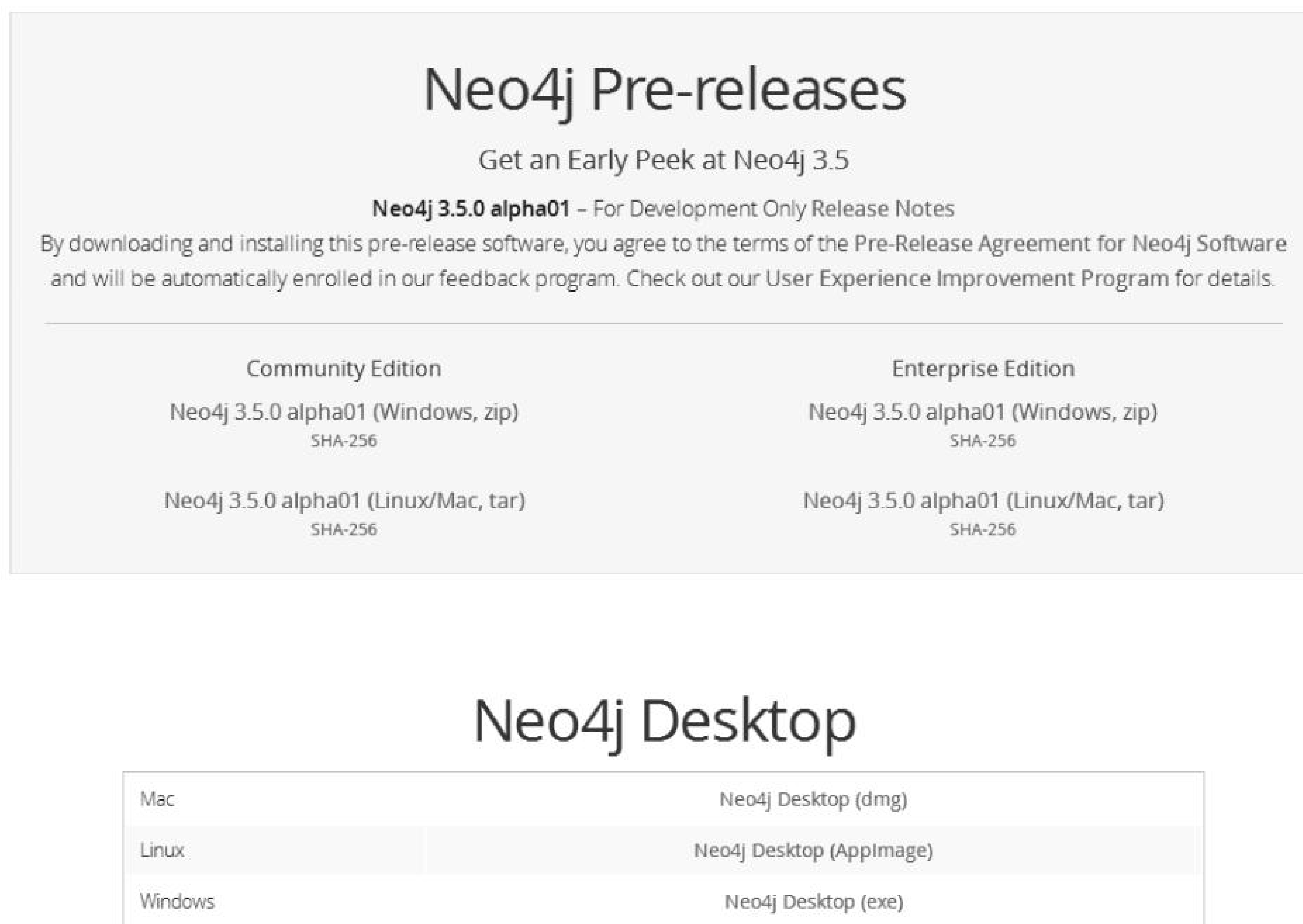


图 8-4 Neo4j 不同版本页面展示图

Community Edition 是社区版，Enterprise Edition 是企业版，Neo4j Desktop 是桌面版，Neo4j Pre-releases 是预发行的最新版。一般来说，下载稳定版本即可，切勿盲目追求最新版。本例下载的是 Mac 版的 Neo4j-community-3.3.2-unix.tar。Neo4j 需要 jdk 8 以上才能运行，所以需要安装 jdk 8。jdk 8 的具体安装方法详见附录 E。

配置好环境以后，在 Mac 下打开 Neo4j 的命令是./neo4j start，如图 8-5 所示。

```
zhaikundeiMac:bin zhaikun$ ./neo4j start
Active database: graph.db
Directories in use:
  home:         /Users/zhaikun/neo4j
  config:       /Users/zhaikun/neo4j/conf
  logs:         /Users/zhaikun/neo4j/logs
  plugins:      /Users/zhaikun/neo4j/plugins
  import:       /Users/zhaikun/neo4j/import
  data:         /Users/zhaikun/neo4j/data
  certificates: /Users/zhaikun/neo4j/certificates
  run:          /Users/zhaikun/neo4j/run
Starting Neo4j.
Started neo4j (pid 1280). It is available at http://localhost:7474/
There may be a short delay until the server is ready.
See /Users/zhaikun/neo4j/logs/neo4j.log for current status.
zhaikundeiMac:bin zhaikun$ 
```

图 8-5 Mac 版 Neo4j 启动界面图

启动成功以后，在浏览器中打开链接地址：http://localhost:7474/，如图 8-6 所示。

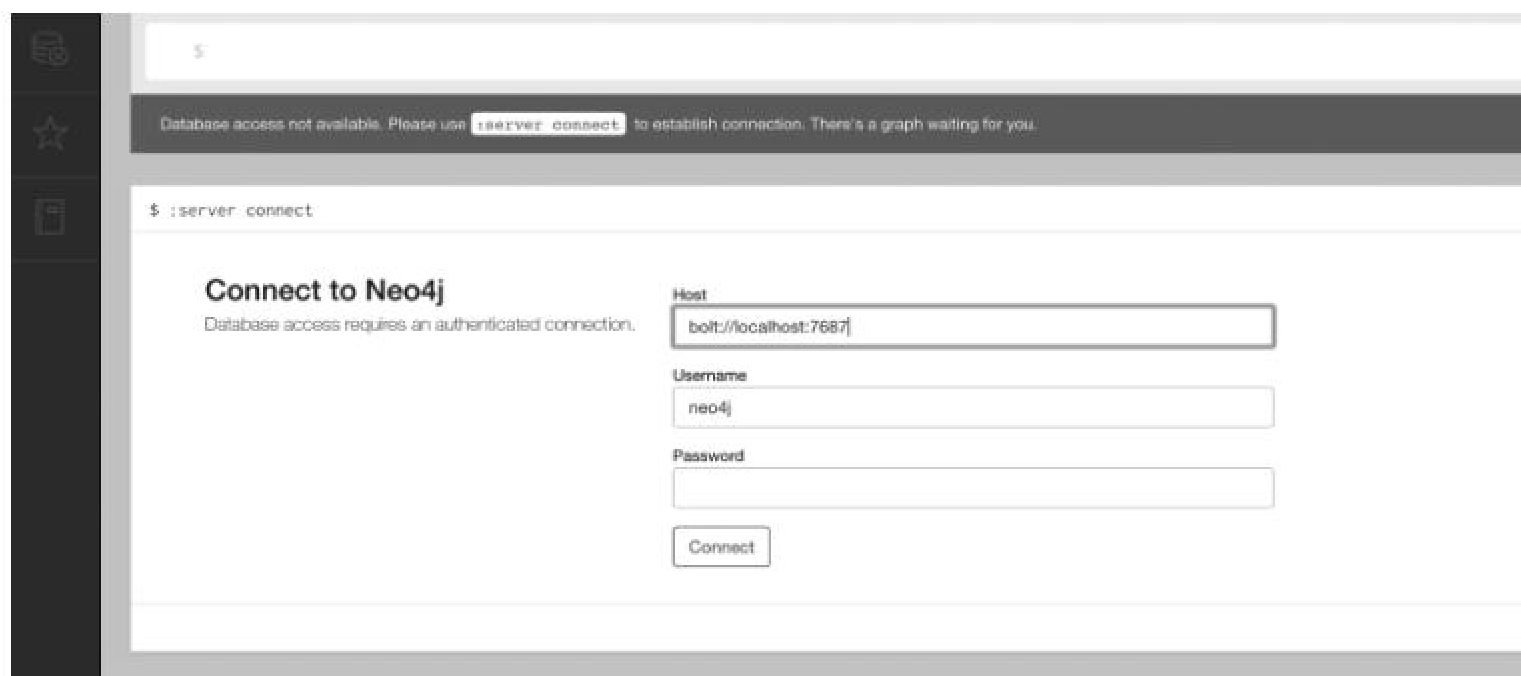


图 8-6　Neo4j 配置连接的主界面图

初始用户名和密码都是 Neo4j，首次打开需要修改密码，如图 8-7 所示。

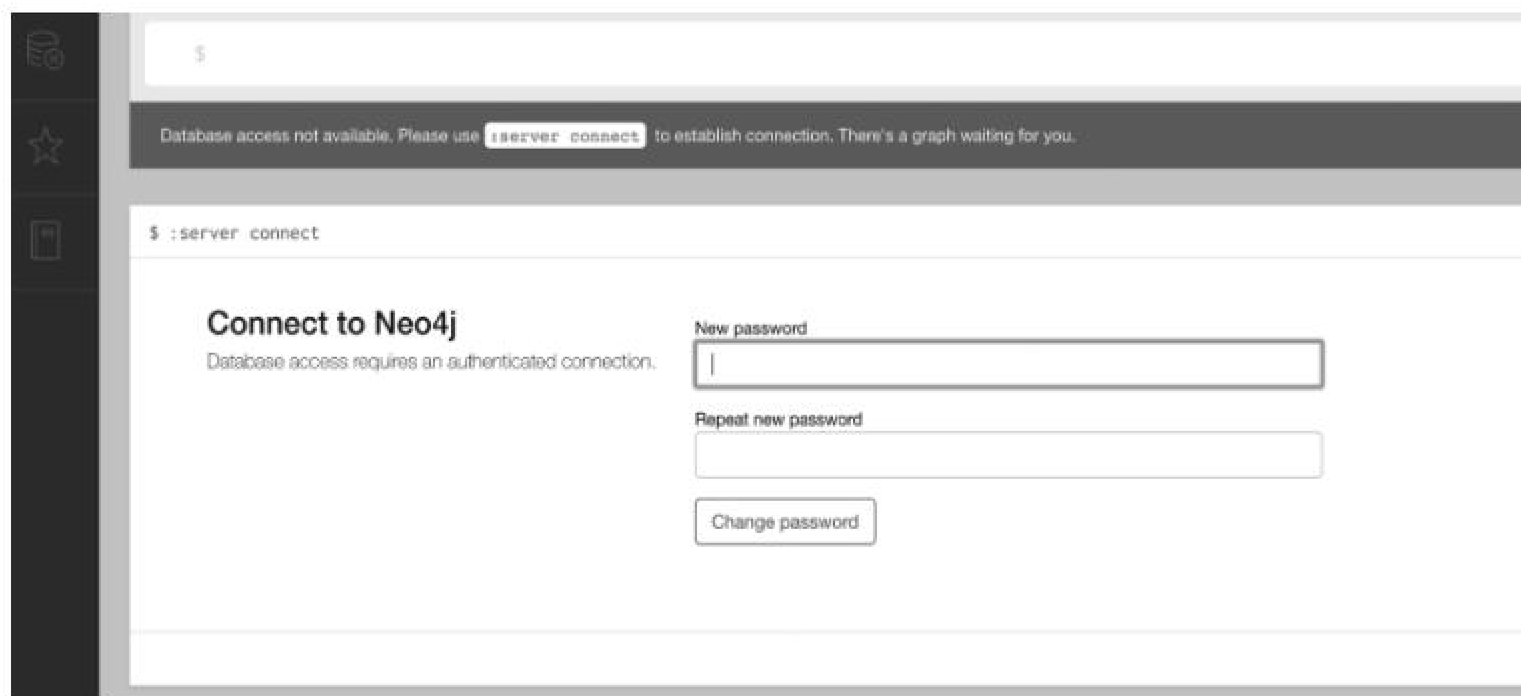


图 8-7　Neo4j 重置密码界面图

重置密码后，就进入了如下界面，如图 8-8 所示。

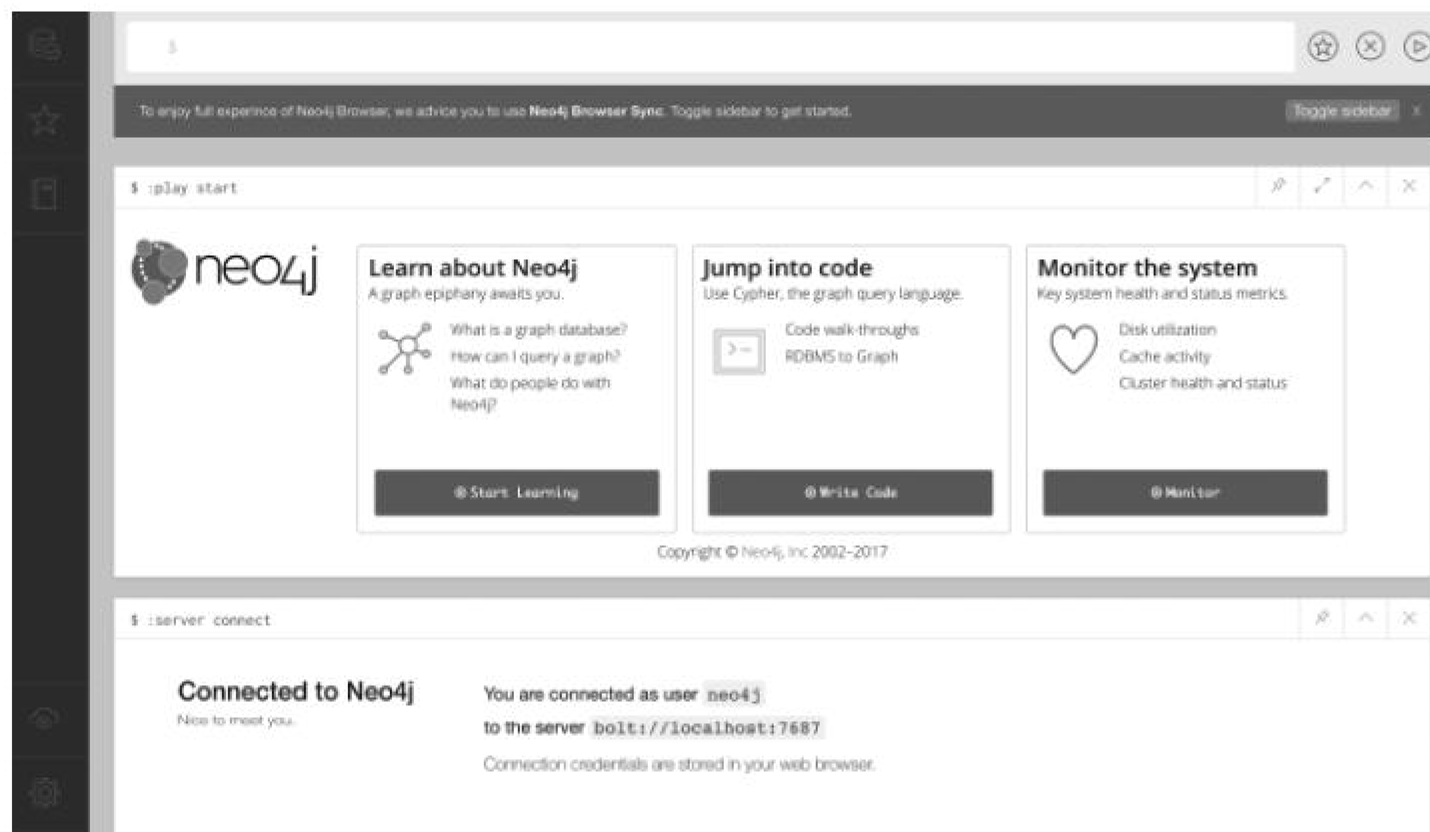


图 8-8　Neo4j 配置连接完成界面图

想要使用 Neo4j 中文版的读者,可以访问 http://www.we-yun.com/,下载中文版的进行练习。

8.2 图形界面介绍

相信大家一定迫不及待地想看看图数据库是个什么样的效果。这边先举个简单的例子:单击中间框 Write Code 按钮,然后单击 create a graph 按钮,会出现 movie graph 案例介绍,往右翻页就是相关代码,运行代码后的结果如图 8-9 所示。

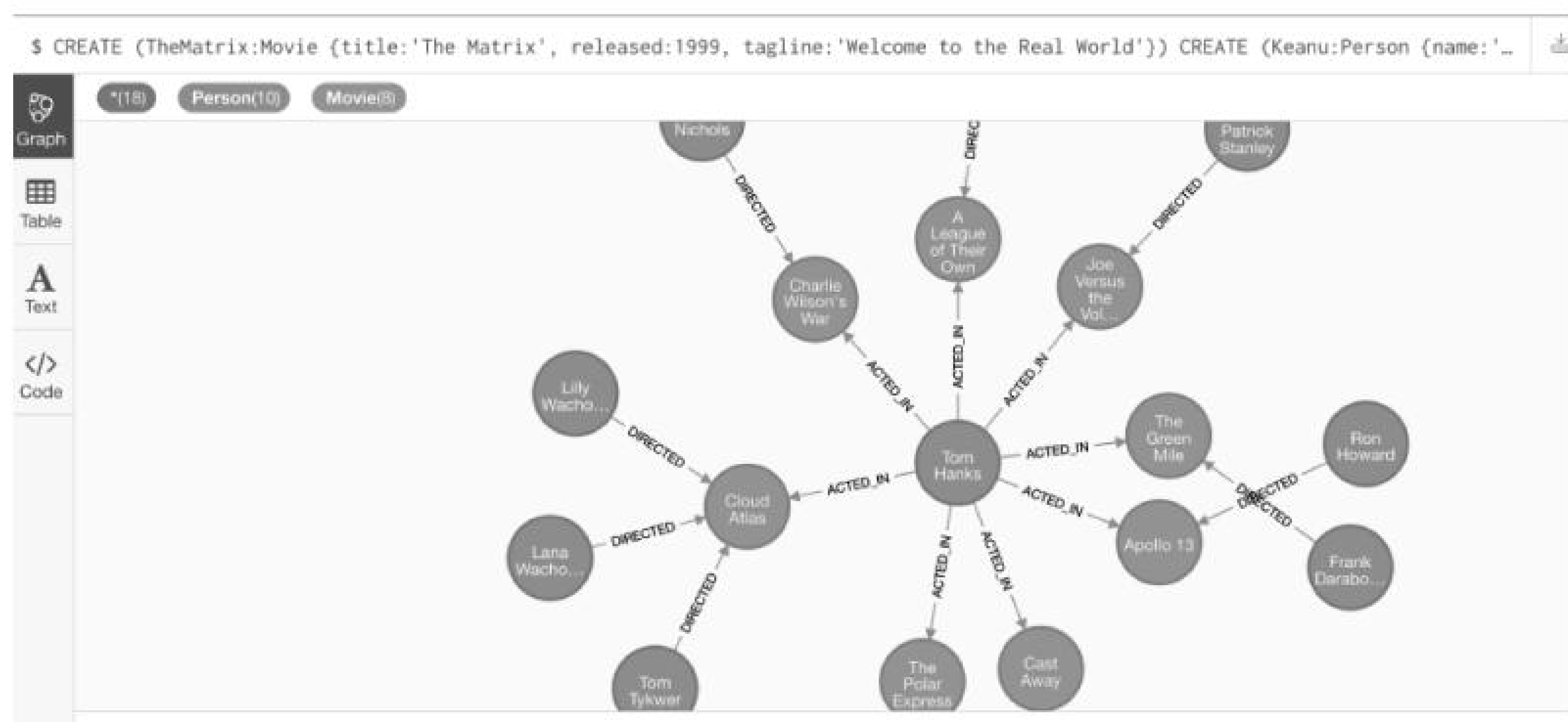


图 8-9 Neo4j 自带图例界面图

由图 8-9 可以看出,每个圆圈代表一个节点,每条边代表着一个关系。左上角蓝色图标"person"表示用蓝色代表"人物",绿色图标"movie"表示用绿色代表"电影"。

单击蓝色人物图标,如图 8-10 所示。

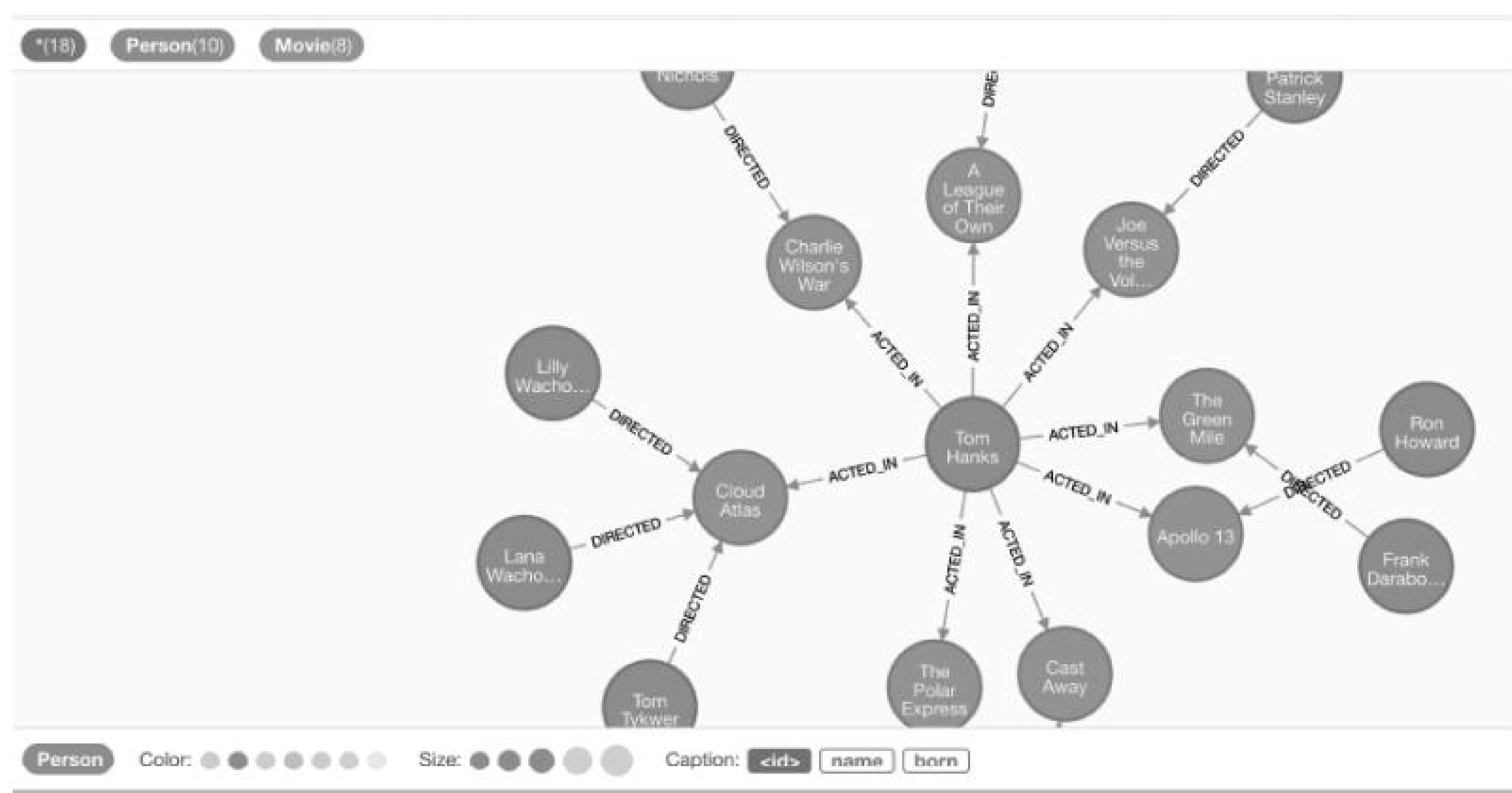


图 8-10 Neo4j 自带图例单击人物图标后的界面图

最下方出现了一行功能栏，表示可以选择颜色、大小和显示的内容。调整图像大小和颜色之后，所得结果如图 8-11 所示。

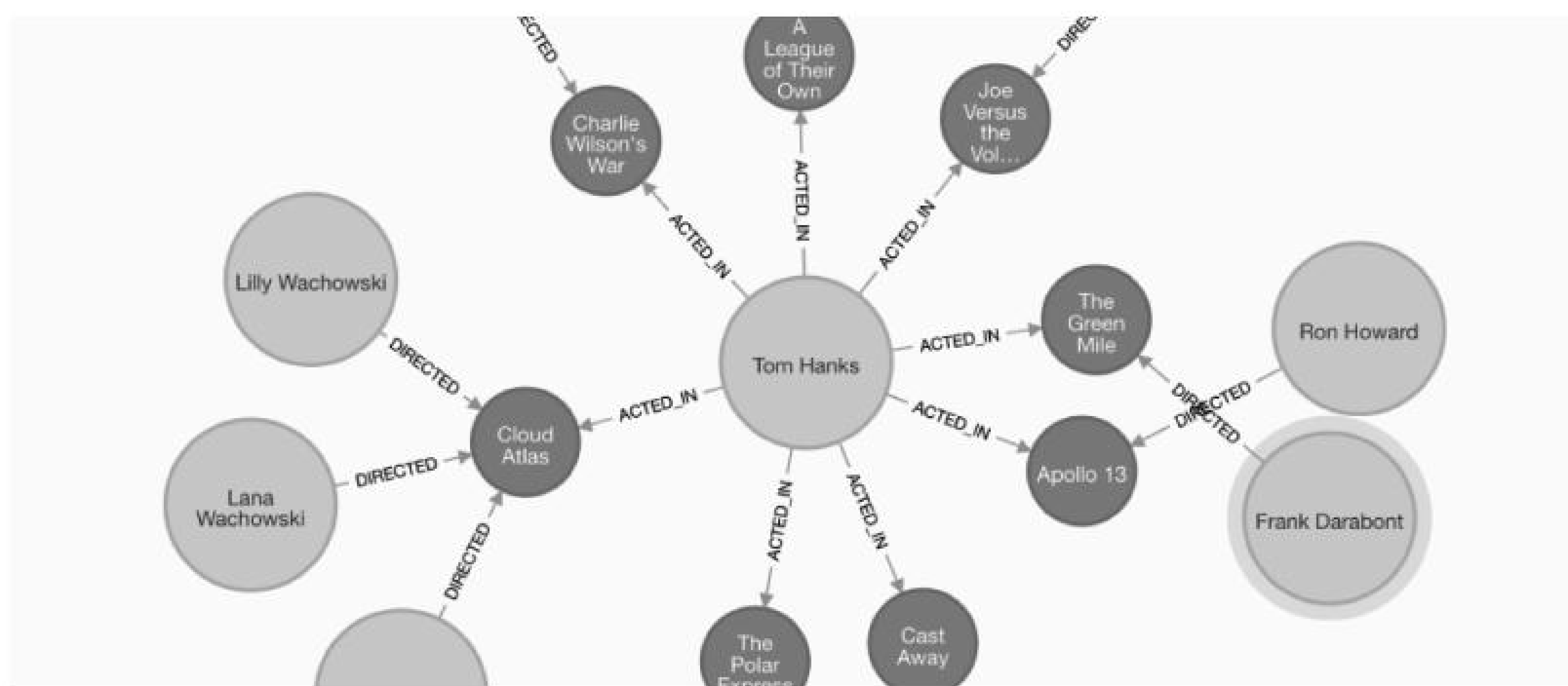


图 8-11　Neo4j 自带图例调整后的界面图

大家会发现人物图标变大了，颜色也变了。这样的功能可以很方便地标识出不同的节点，更加直观易懂。

单击节点，如图 8-12 所示。

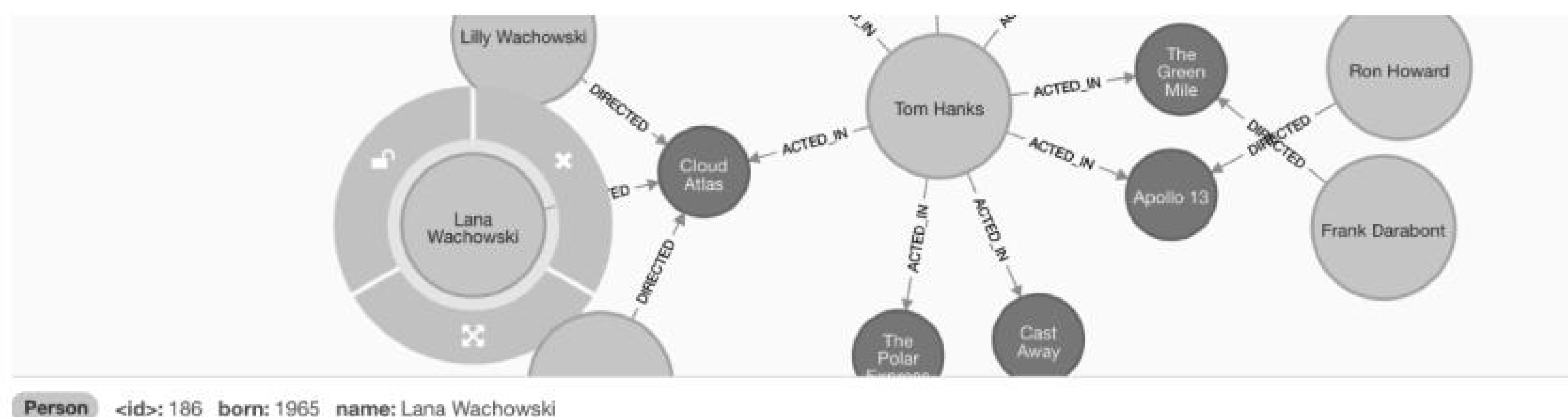


图 8-12　Neo4j 自带图例单击节点后的界面图

这时，下面的功能区会显示节点的属性。

同理，单击关系，下面的功能区也会显示关系的属性。如图 8-13 所示。

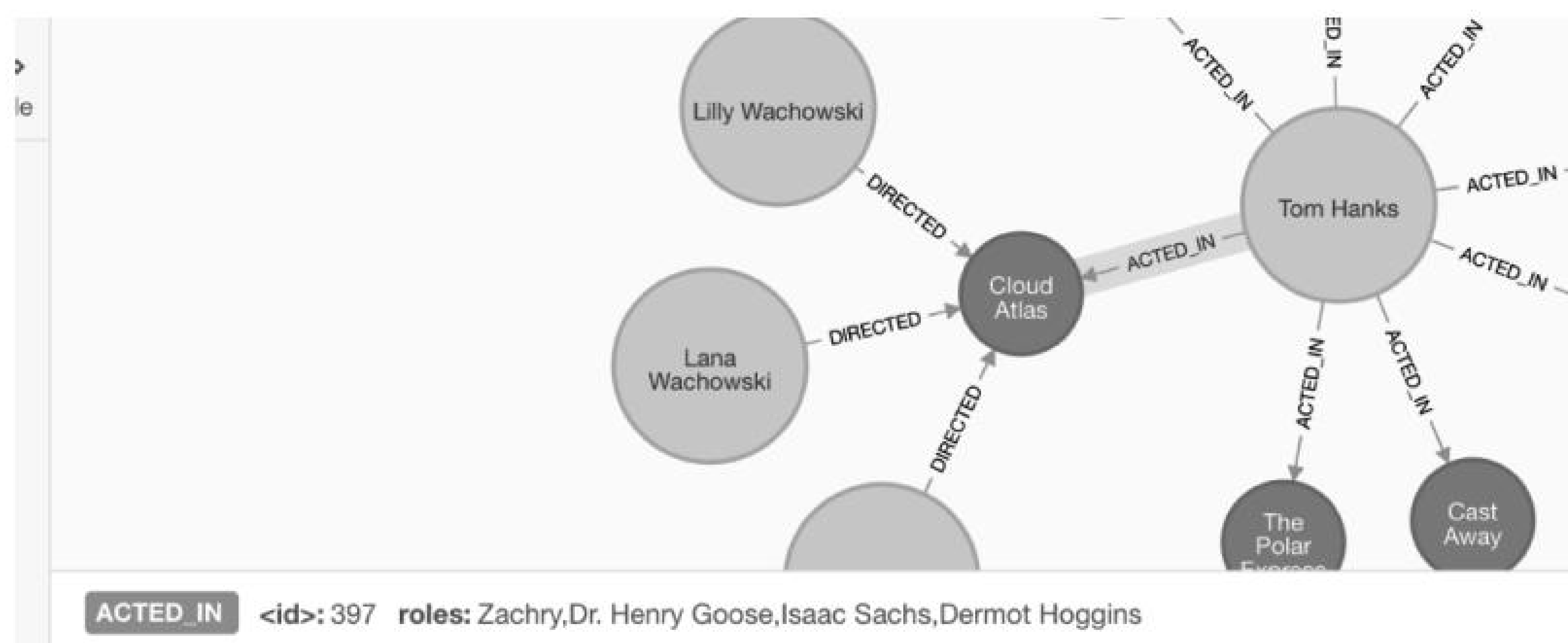


图 8-13　Neo4j 自带图例单击关系后的界面图

8.3 Cypher 语言

Neo4j 是一个 NoSQL 数据库,用的查询语言是 Cypher 语言,与 SQL 语句非常相似。本节将介绍 Cypher 的基本语法。

Cypher 是一种声明式图查询语言,具有语法简单、功能强大等优点。通常,学习过 SQL 语句的读者可以很轻易地学会 Cypher 语言的用法。

1. Cypher 语言常用关键词语法

Cypher 语言常用的关键词有 4 个,其语法的详细教程可在 Neo4j 客户端查看。当 Neo4j 连接配置完成后,在界面最上方输入":help cypher",如图 8-14 所示。Cypher 语言各关键词对应语法如图 8-15 所示。

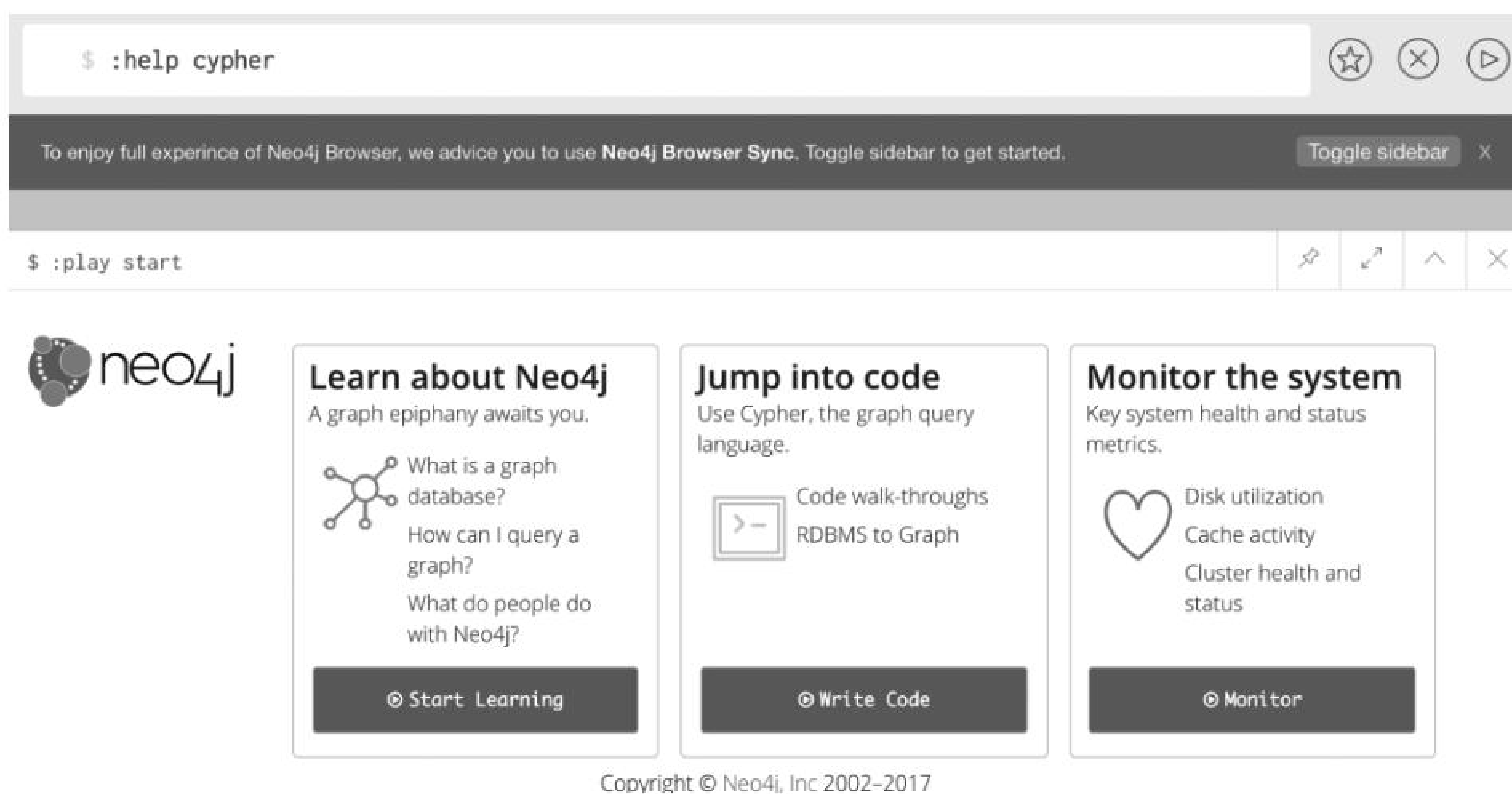


图 8-14 Neo4j 中输入":help cypher"的界面图

2. Cypher 语言常用关键词的作用与用法

(1) CREATE 的作用——创建

创建节点:

```
CREATE (n {name: $ value})
```

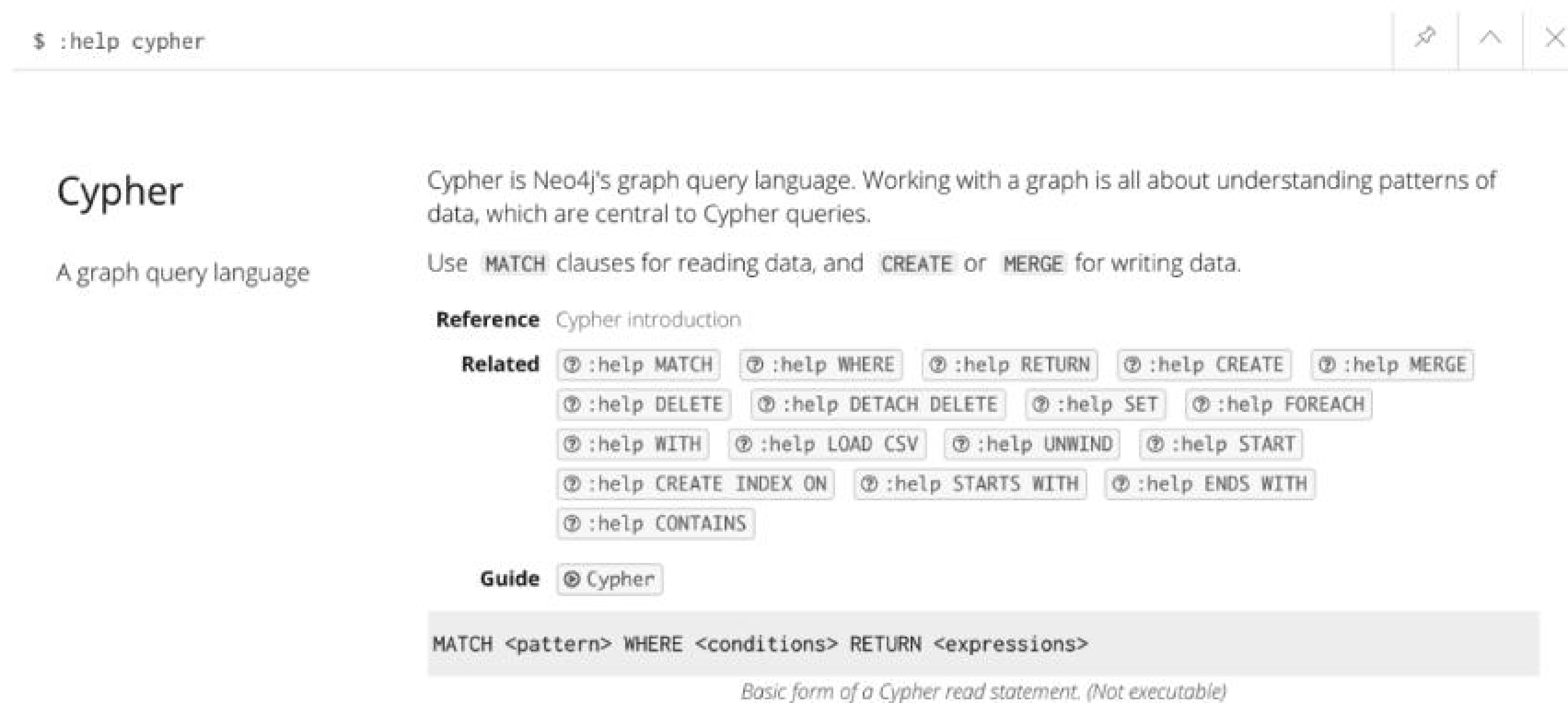


图 8-15　Cypher 语言各关键词对应语法信息界面图

创建关系：

```
CREATE (n)-[r:KNOWS]->(m)
CREATE (n)-[:LOVES {since: $value}]->(m)
```

（2）MATCH 的作用——查询匹配

查询节点：

```
MATCH (n:Person)-[:KNOWS]->(m:Person)
WHERE n.name = 'Alice'
```

查询路径：

```
MATCH p = (n)-->(m)
```

查询具有属性的节点：

```
MATCH (n {name: 'Alice'})-->(m)
```

（3）WHERE 的作用——条件选择

```
WHERE n.property = 100
```

（4）LOAD CSV 的作用——导入文件

```
LOAD CSV WITH HEADERS FROM
'file:///test.csv' as line
CREATE (:Artist {name: line.Name, year:toInteger(line.Year)})
```

8.4 Neo4j 案例 1——《天龙八部》的人物关系分析

本节将以《天龙八部》这本小说为例，讲解一下 Neo4j 的具体使用方法。

```
'''导入文件'''
using periodic commit 1000
load csv with headers from 'file:///tianlongbabu.csv' as line
merge (n1:人物 {姓名:line.Source})              //创建源节点
merge (n2:人物 {姓名:line.Target})              //创建目标节点
merge (n1) - [r:相识] ->(n2)                    //创建关系
on create set r.weight = toInt(line.Weight)    //添加关系属性 weight
```

将天龙八部数据的文件放在 Neo4j 文件夹下的 import 文件夹里，如图 8-16 和图 8-17所示。

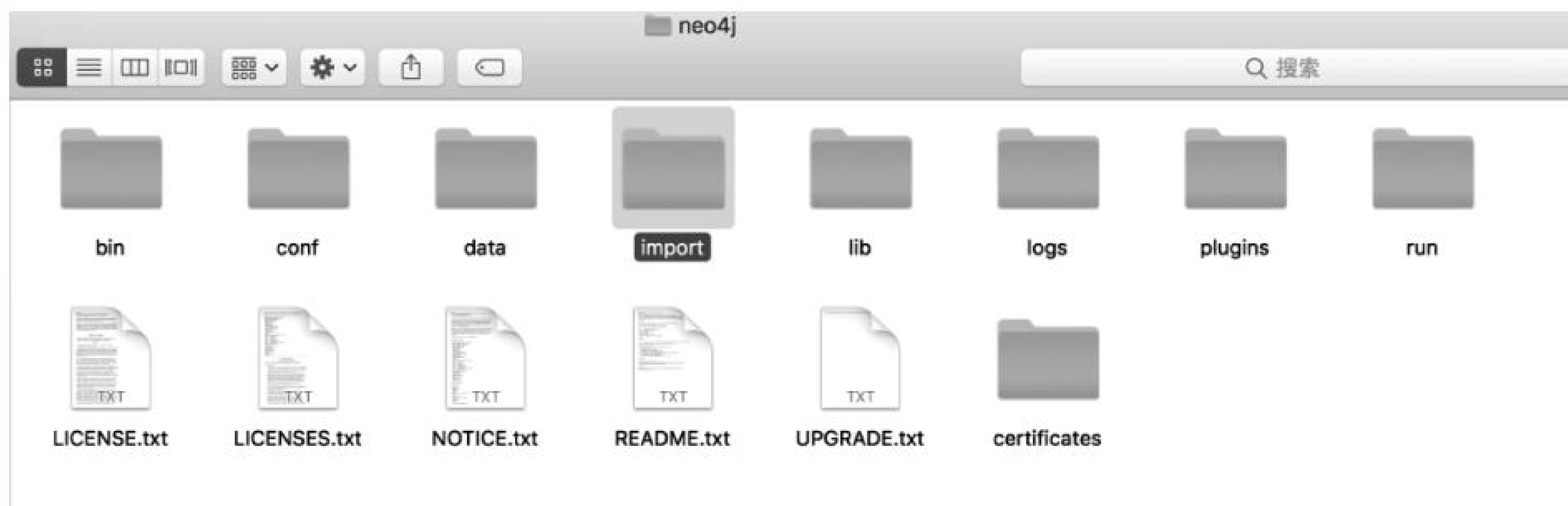


图 8-16　Neo4j 的文件结构展示图



图 8-17　天龙八部数据放在 import 文件夹下的展示图

在浏览器中打开链接地址：http//localhost:7474/，单击中间框 Write Code 按钮，输入如图 8-18 所示的代码，导入文件。

导入结果为导入了 1164 个标签、1164 个节点，设置了 9934 个属性，创建了 8770

个边。



```
using periodic commit 1000
load csv with headers from 'file:///tianlongbabu.csv' as line
merge (n1:人物 {姓名:line.Source})        //创建源节点
merge (n2:人物 {姓名:line.Target})        //创建目标节点
merge (n1)-[r:相识]->(n2)                 //创建关系
on create set r.weight = toInt(line.Weight)  //添加关系属性weight
```

sing periodic commit 1000 load csv with headers from 'file:///tianlongbabu.csv' as line merg…

Added 1164 labels, created 1164 nodes, set 9934 properties, created 8770 relationships, completed after 2378 ms.

图 8-18　导入数据之后的结果展示图

下面预览一部分数据，让大家有个直观的印象，如图 8-19 所示。

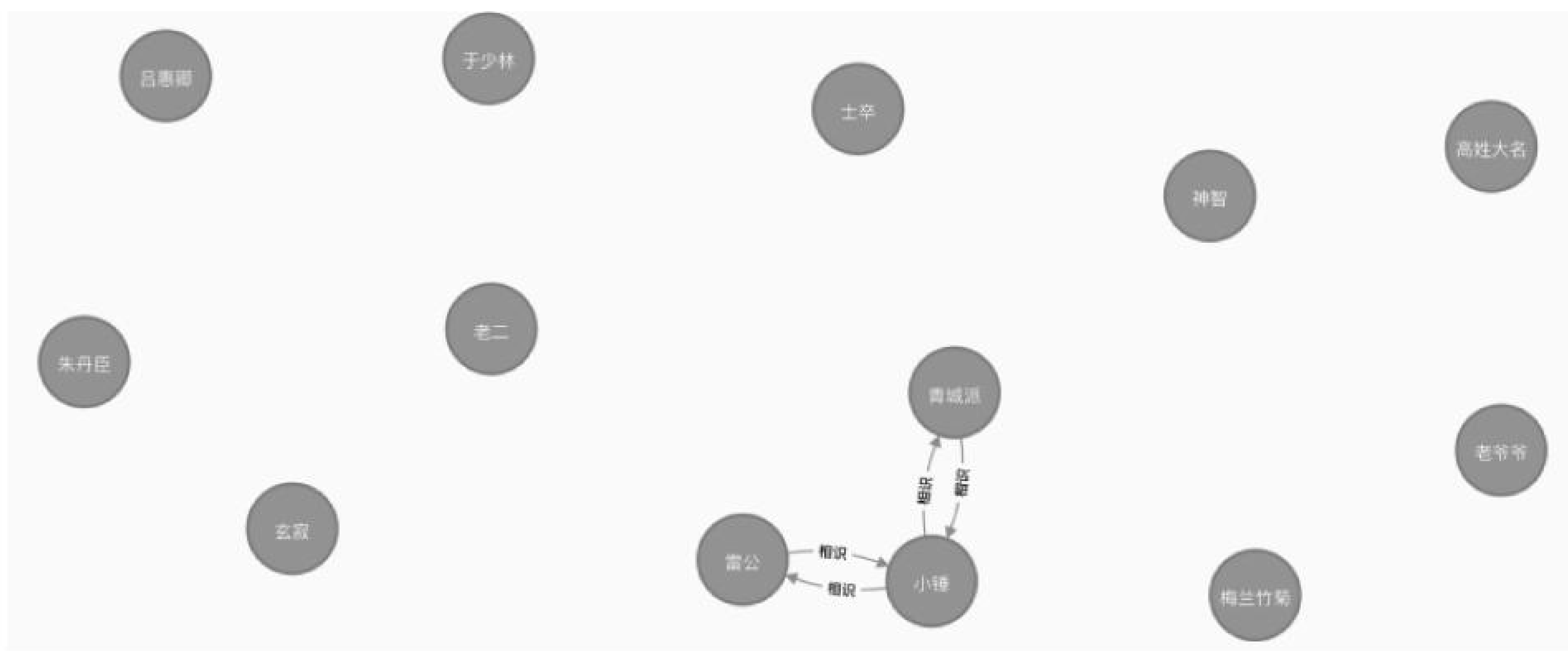


图 8-19　导入数据之后的节点预览展示图

图数据库最重要的应用就是进行社交网络分析，应用如下。

一度人脉：

```
match p = (n1{姓名:'木婉清'}) - [ * ..1] - (n2{姓名:'阿朱'}) return p limit 10
```

运行结果如图 8-20 所示。说明木婉清与阿朱没有直接认识。

(no changes, no records)

图 8-20　木婉清与阿朱的一度人脉结果图

二度人脉。

```
match p = (n1{姓名:'木婉清'}) - [ * ..2] - (n2{姓名:'阿朱'}) return p limit 10
```

运行结果如图 8-21 所示。

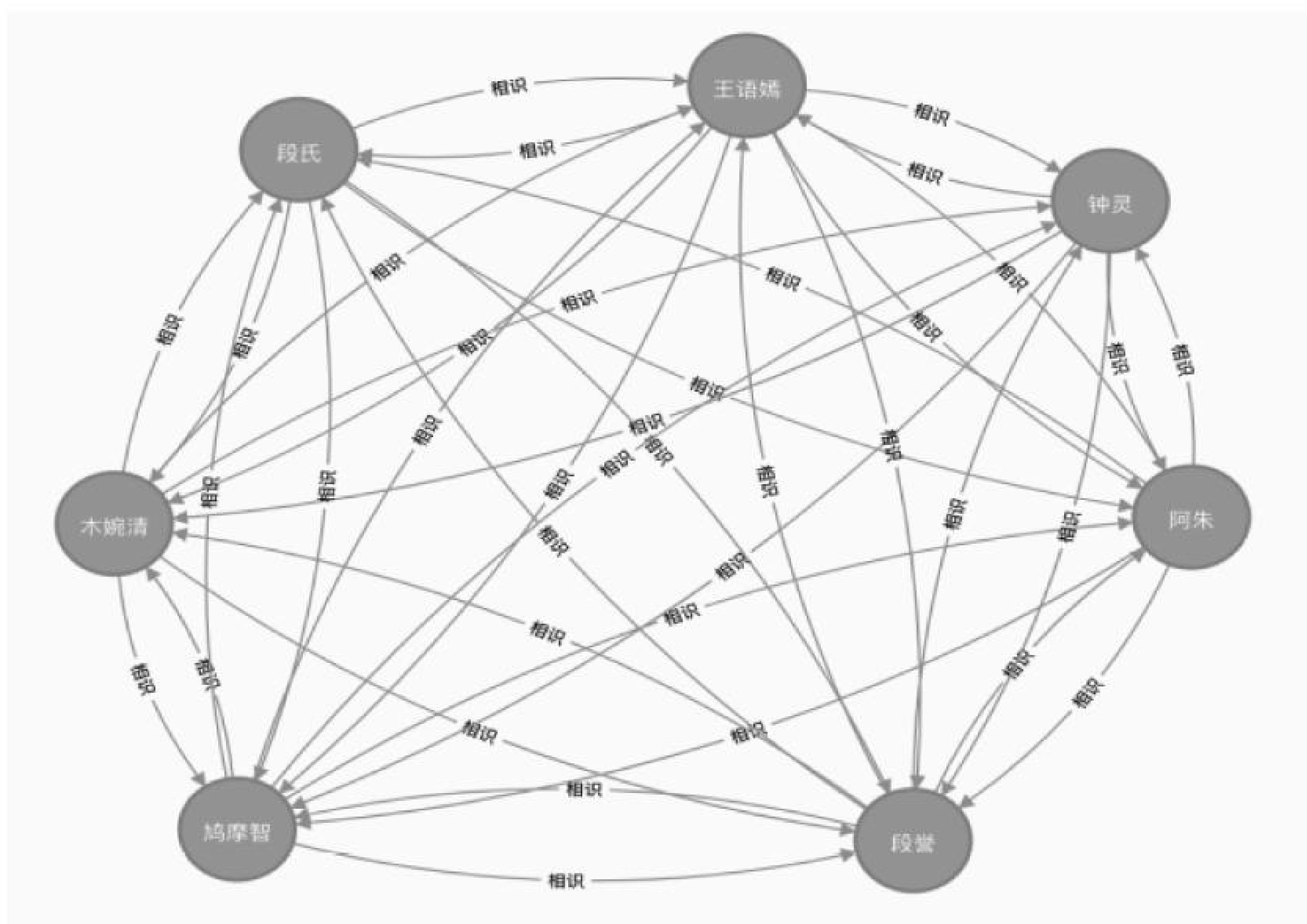


图 8-21 木婉清与阿朱的二度人脉结果图

说明木婉清想认识阿朱可以通过“段氏”“王语嫣”“钟灵”等人。金融上常用的是一度和二度人脉，再扩展下去，其实际意义不是很大，感兴趣的读者可以继续扩展。

最短路径：

```
match p = shortestpath((n1{姓名:'木婉清'}) - [ * ..6] - (n2{姓名:'阿朱'}) ) return p
```

运行结果如图 8-22 所示。

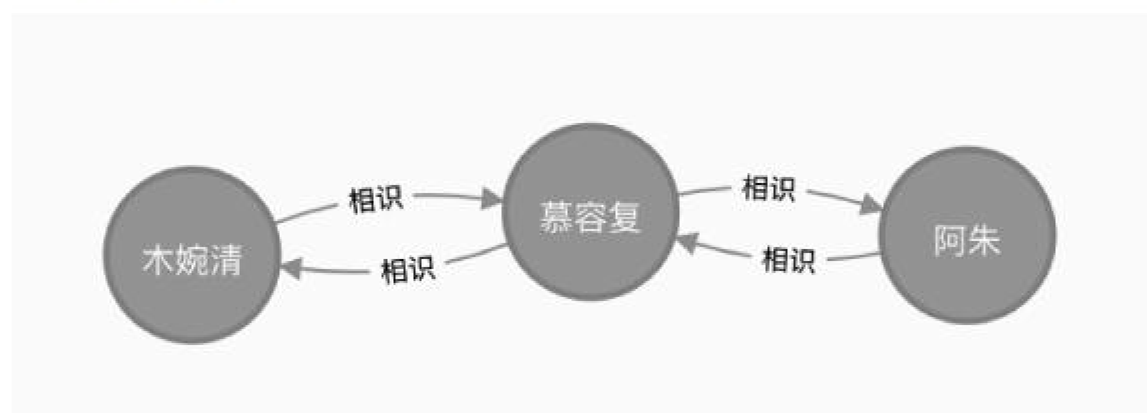


图 8-22 木婉清与阿朱认识的最短路径结果图

这是六度人脉范围内，木婉清希望认识阿朱的任意一条最短路径。

度中心性：就是节点的连接边数。

```
match (n:人物)
return n.姓名 as 姓名, size((n)-[]-()) as degree
order by degree desc
limit 10
```

运行结果如图 8-23 所示。

姓名	degree
"段誉"	600
"萧峰"	428
"乔峰"	310
"慕容复"	266
"阿朱"	246
"王语嫣"	236
"段正淳"	236
"木婉清"	228
"鸠摩智"	222
"少林寺"	200

图 8-23 天龙八部中的度中心性结果排序展示图

结果很有意思，段誉的关系数最多。那么谁与段誉的关系最亲密呢？代码如下：

```
match (n1:人物{姓名:'段誉'})-[r]->(n2)
return n1.姓名 as 源, n2.姓名 as 目标, r.weight as 关系权重
order by 关系权重 desc
limit 10
```

运行结果如图 8-24 所示。

源	目标	关系权重
"段誉"	"王语嫣"	568
"段誉"	"木婉清"	469
"段誉"	"慕容复"	259
"段誉"	"钟灵"	220
"段誉"	"鸠摩智"	198
"段誉"	"王姑娘"	130
"段誉"	"段公子"	121
"段誉"	"阿朱"	119
"段誉"	"段正淳"	113
"段誉"	"萧峰"	110

图 8-24 天龙八部中与段誉关系最亲密排序展示图

结果不出所料,段誉还是很有妹子缘的,并且王语嫣遥遥领先。

8.5 Neo4j 案例 2——金融场景中的社交网络分析

本节将用工业上常用的 Neo4j 数据库进行社交网络与反欺诈分析。

先用 admin import 语句导入节点 csv 文件和关系 csv 文件。因为此文件涉及敏感信息,所以不会上传。那么大家在尝试的时候,可以根据文件格式,编造类似数据,进行文件导入。

先看看坏客户与其相关联的亲友界面图,如图 8-25 所示。

语句为:

```
MATCH p = (n1:坏客户) -->(n2:坏客户亲友) RETURN p LIMIT 25
```



图 8-25 坏客户与其相关联的亲友图谱

好客户与其相关联的亲友图谱,如图 8-26 所示。

语句为:

```
MATCH p = (n1:好客户) -->(n2:好客户亲友) RETURN p LIMIT 25
```

善于观察的读者可以发现,图 8-25 中正中间的圆圈代表的是坏客户,周边的圆圈代表的是坏客户的联系人;图 8-26 中正中间的圆圈代表的是好客户,周边的圆圈



图 8-26　好客户与其相关联的亲友图谱

代表的是好客户的联系人。

接下来看看二度人脉。

先看看坏客户与坏客户之间有没有连接呢？坏客户与坏客户之间的图谱如图 8-27所示。

```
match p = (n1:坏客户) - [ * ..2] - (n2:坏客户) return p limit 25
```

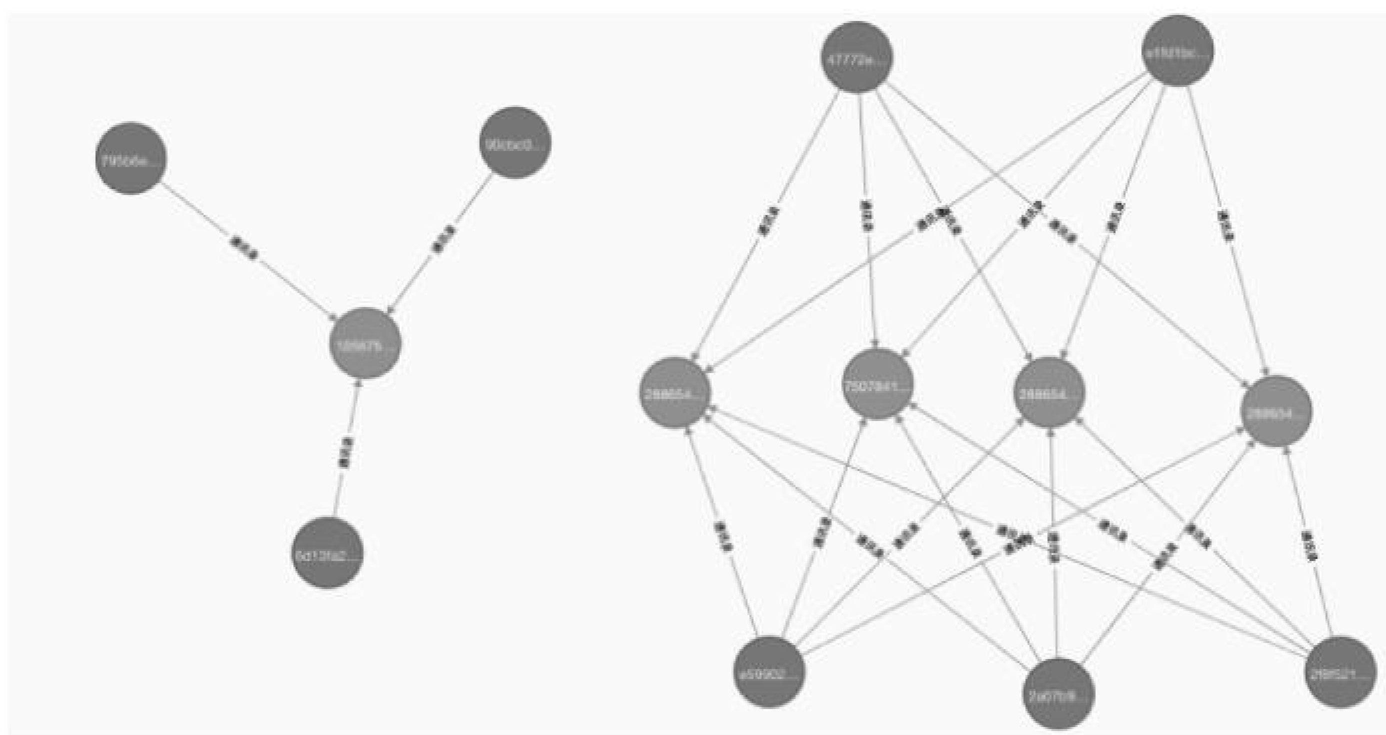


图 8-27　坏客户与坏客户之间的图谱

从图 8-27 中，可以发现坏客户与坏客户之间是有连接的。仔细一看，相关联的电话号码还很类似，如果这些号码是黑中介的号码，那么一般情况下，这些客户就是欺诈客户，可以拒绝他们的贷款申请。

再来看看好客户与好客户之间有没有连接吧。好客户与好客户之间的图谱如

图 8-28所示。

```
match p = (n1:好客户) -[ * ..6] -(n2:好客户) return p limit 25
```

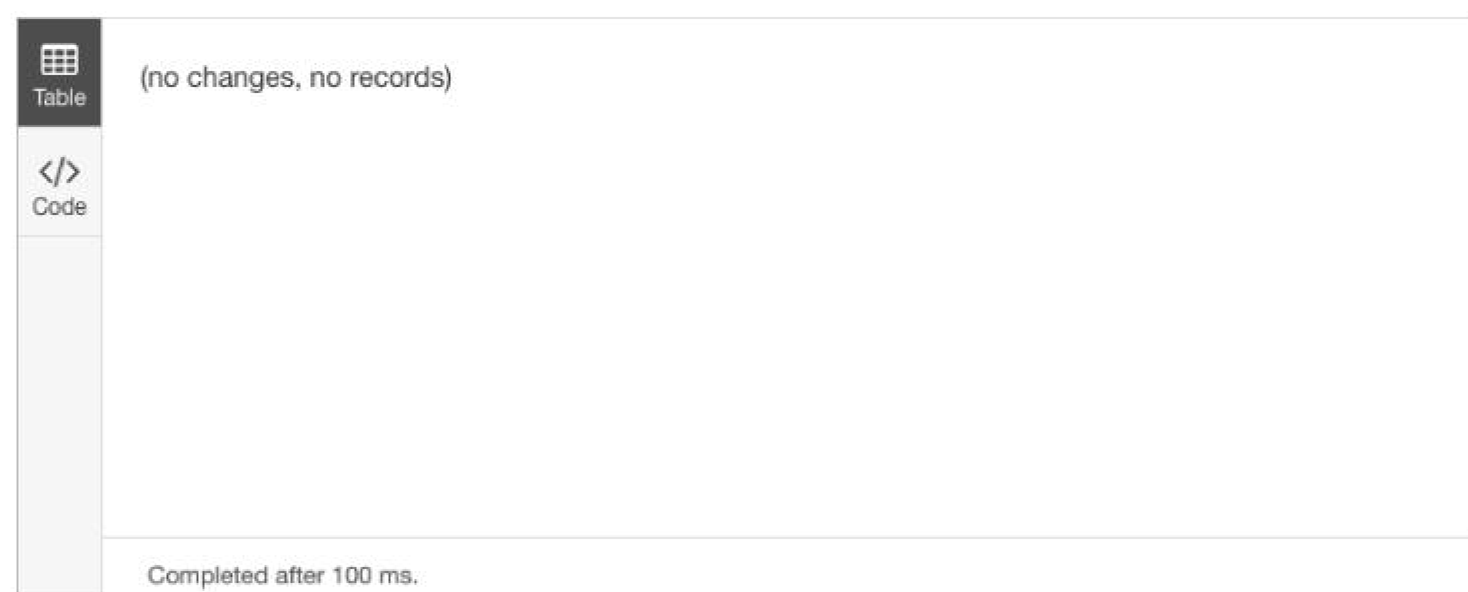


图 8-28　好客户与好客户之间的图谱

六度人脉内都没有连接,看来没有这么一个人同时连接了多个正常客户。

接下来,再分析 3 个指标吧。

(1) 度中心性

```
match (n:好客户)
return n.user_id as id, size((n) -[:通讯录] -()) as 度中心性, labels(n) as 标签
order by 度中心性 desc
```

运行结果如图 8-29 所示。

id	度中心性	标签
"f88a5539983543e5a88f57a8668720e8"	772	["好客户"]
"f48dc6a67b1d4f66b6511d63e2deadf1"	723	["好客户"]
"fb28ad2735b44191a2e9fd4e51f94001"	204	["好客户"]
"4b8af0e28ea84323b4923ebb68545486"	172	["好客户"]
"a45c2e3738e348a59612f40eb8480d0c"	137	["好客户"]
"879c9769a27542c391036cf47b2a693c"	123	["好客户"]
"87dcde927e9d460599488926cbad38b2"	121	["好客户"]
"c27d6537ebce49d9b0777bb24045f956"	118	["好客户"]
"11018701134d41b7beb351e3cabb912c"	117	["好客户"]
"c68bafdfb1be4a0f921f62c70ad5e98e"	117	["好客户"]
"5487e8751773485cb0c149e455c062ae"	87	["好客户"]
"d155a1e0cade4f34ba16d1948b88fdd9"	74	["好客户"]
"72a6e71233454e9c96accfe2c0cdb6e6"	70	["好客户"]
"cb9457e3a9084b329ef5a67c16d4840f"	51	["好客户"]
"ef74a8cb9aa640ca97d0b71e4cf9e159"	47	["好客户"]
"2576bbd4427d442fa3293f2be769a760"	24	["好客户"]

图 8-29　好客户的度中心性排序图

（2）接近中心性

```
match (n1)
with collect(n1) as nodes
call apoc.algo.closeness(['通讯录'], nodes, 'both') yield node, score
return node.user_id as id, score as 接近中心性
order by 接近中心性 desc limit 100
```

运行结果如图 8-30 所示。

id	接近中心性
"10e072d66a5a4f92a7d385aeeb9cceb2"	0.1
"8cdf08b21bba46d4a411b9dc4b1e933a"	0.1
"f615f73c8e7f4d26b0283a0c4d3b693d"	0.07692307692307693
"0a9ee82e15e8473ca4fc432dbc822d42"	0.07142857142857142
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842
null	0.05263157894736842

图 8-30　客户的接近中心性排序图

（3）中介中心性

```
match (n1)
with collect(n1) as nodes
call apoc.algo.betweenness(['通讯录'], nodes, 'both') yield node,score
set node.betweenness = score
return node.user_id as id, score as 中介中心性
order by 中介中心性 desc limit 100
```

运行结果如图 8-31 所示。

下面把以上指标最高的节点挑出来，看看他们的连接状态。

为了让大家有更直观的印象，下面选度中心性为 24 的节点，看看它的连接情况。

运行代码如下：

```
match p = (n1{user_id:'2576bbd4427d442fa3293f2be769a760'}) -[] ->(n2) return p
```

运行结果如图 8-32 所示。

由图 8-32 可知，这个节点的连接边数就是 24。

id	中介中心性
"e59902bf0bd94f378913451c2ea650bc"	833759.2
"e1fd1bc45f6e4b908f327b94d281edb9"	667349.2
"0b9643dcdb4543738069f91a38b0ffc3"	506521
"f88a5539983543e5a88f57a8668720e8"	297606
"f48dc6a67b1d4f66b6511d63e2deadf1"	261003
null	194094
null	194094
null	194094
null	194094
"47772e023725426e9a1fe2c4920940a2"	192833.2
"6d13fa2ece9a4cc2a73905233835c3b5"	155991

图 8-31　客户的中介中心性排序图

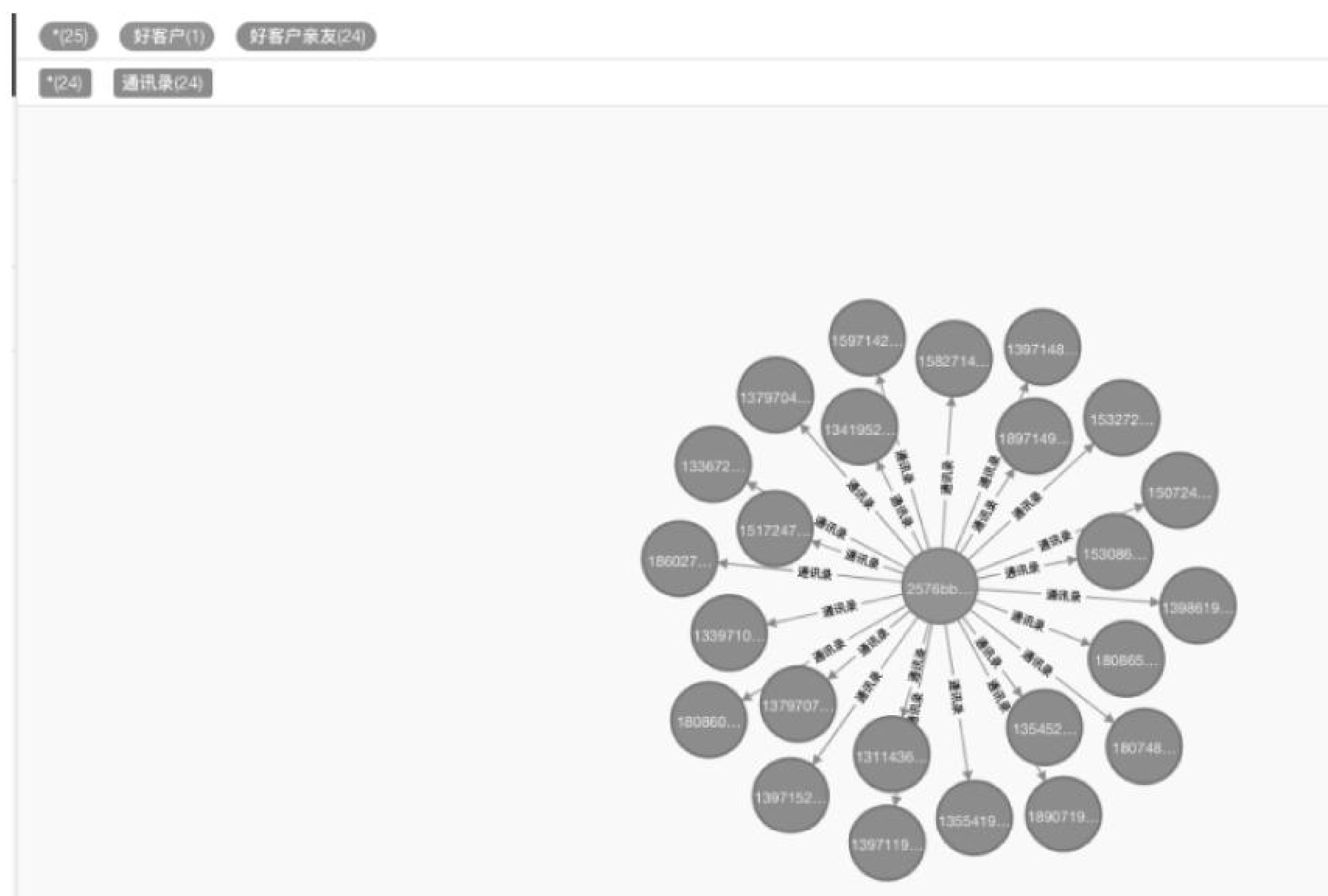


图 8-32　度中心性为 24 的节点关系图谱

Neo4j 的语法与应用场景非常多,还能用于坏客户社区探索,交易欺诈判定、供应链金融等。

8.6　Py2neo

Neo4j 是一个非常好用的图关系数据库,它可以支持 Python 驱动,一般用 Py2neo,安装语句为:pip install py2neo,具体使用方法官网上有非常详细的解释,感

兴趣的读者可以参考官网地址：http://py2neo.org/v3/index.html，如图 8-33 所示。安装步骤详见附录 F。

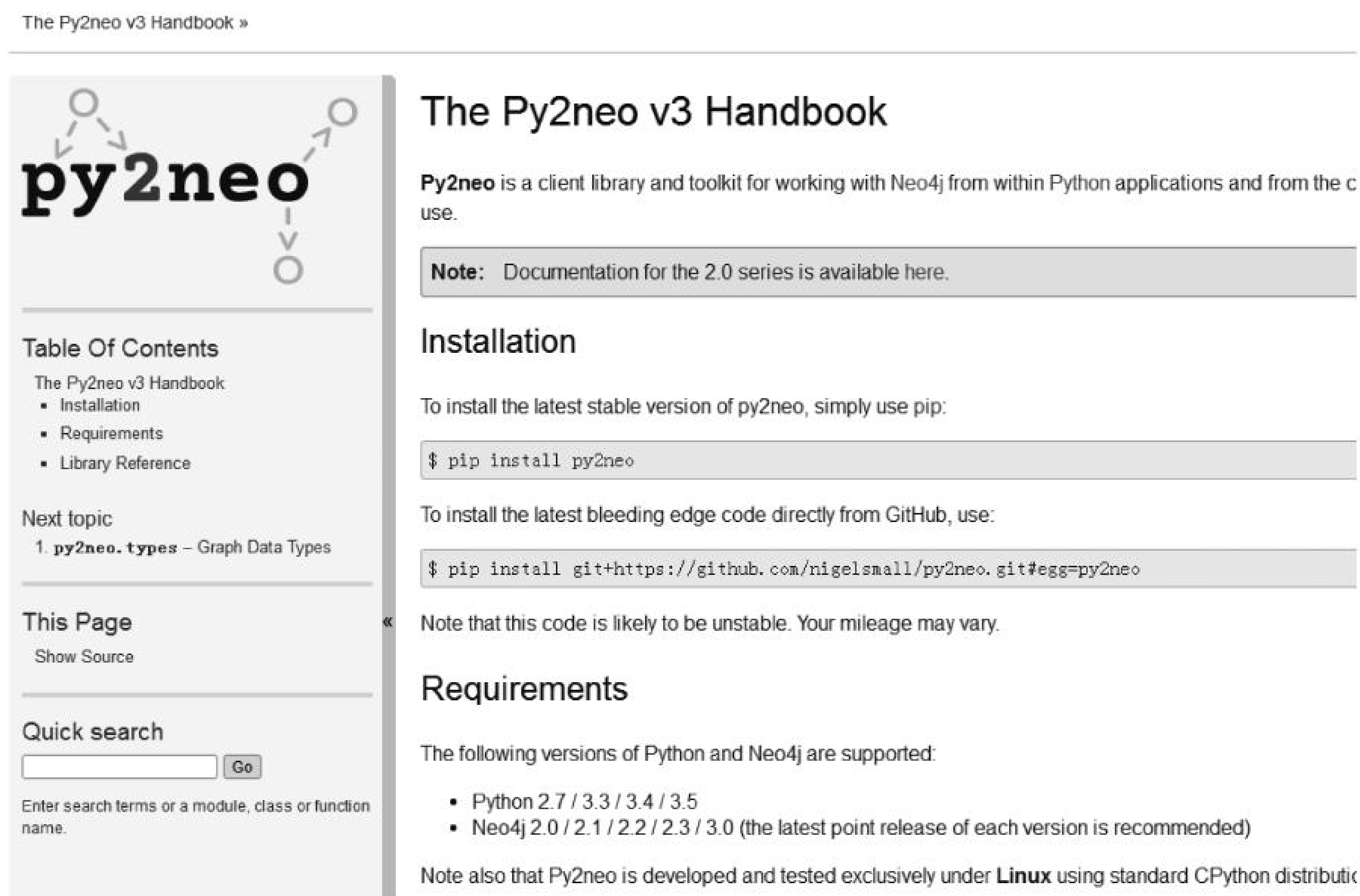


图 8-33 Py2neo 官网指导手册展示图

在完成安装之后，在 Python 中调用 py2neo 即可，常用的两个数据结构就是节点和关系，即 Node 和 Relationship。

```
from py2neo import Graph, Node, Relationship
```

连接 Neo4j 的方法很简单，代码如下：

```
test_graph = Graph(
    "http://localhost:7474",
    username = "neo4j",
    password = "neo4j"
)
```

test_graph 就是已建立好的 Neo4j 的连接。在连接 Neo4j 数据库之后，就可以创建节点了。节点的建立要用到 py2neo.Node，建立节点的时候要定义它的节点类型(label)以及一个基本属性(property，包括 property_key 和 property_value)。以下代码为建立了两个测试节点。

```
test_node_1 = Node(label = "Person", name = "test_node_1")
test_node_2 = Node(label = "Person", name = "test_node_2")
test_graph.create(test_node_1)
test_graph.create(test_node_2)
```

这两个节点的类型(label)都是 Person,而且都具有属性(property_key)为 name,属性值(property_value)分别为"test_node_1","test_node_2"。

创建好节点之后,就可以创建节点间的关系(Relationship)了。由于节点关系是有向的,所以在建立关系时,必须定义一个起始节点和一个结束节点。值得注意的是,起始节点可以和结束节点是同一个点,这时候该节点就存在一个指向自己的关系。

```
node_1_call_node_2 = Relationship(test_node_1, 'CALL', test_node_2)
node_1_call_node_2['count'] = 1
node_2_call_node_1 = Relationship(test_node_2, 'CALL', test_node_1)
node_2_call_node_1['count'] = 2
test_graph.create(node_1_call_node_2)
test_graph.create(node_2_call_node_1)
```

如以上代码,分别建立了 test_node_1 指向 test_node_2 和 test_node_2 指向 test_node_1 两条关系,关系的类型为"CALL",两条关系都有属性 count,且值为 1。

在这里有必要提一下,当建立关系时,如果起始节点或者结束节点不存在,则在建立关系的同时建立这个节点。

感兴趣的读者可以参考官方文档进行属性、关系的更新等操作,这里就不再赘述。

8.7 小结

本章主要介绍了 Neo4j 数据库、Cypher 语言,针对 Neo4j 数据库的操作,以两个案例进行讲解,最后介绍了 Py2neo 对 Neo4j 进行相关操作。由于 Neo4j 数据库应用场景非常多,那么在后续的学习中,考验读者的还是查找资料、理解消化能力。

参考文献

[1] David Beazley，Brian K Jones. Python Cookbook(第 3 版)中文版[M]. 陈舸，译. 北京：人民邮电出版社，2015.

[2] Mamdouh Refaat. 信用风险评分卡研究：基于 SAS 的开发与实施[M]. 王松奇，林治乾，译. 北京：社会科学文献出版社，2013.

[3] Raymond Anderson. 信用评分工具：自动化信用管理的理论与实践[M]. 李志勇，译. 北京：中国金融出版社，2017.

[4] Wes McKinney. 利用 Python 进行数据分析[M]. 唐学韬，等译. 北京：机械工业出版社，2014.

[5] 陈建. 信用评分模型技术与应用[M]. 北京：中国财政经济出版社，2005.

附录 A

PyCharm安装步骤

PyCharm 官网链接地址为：http://www.jetbrains.com/pycharm，下载链接地址为：http://www.jetbrains.com/pycharm/download/previous.html。

下面以 PyCharm Community 为例，进行安装说明。

(1) 右击“安装包”，单击“以管理员身份运行”。

(2) 单击 Next 按钮，如图 A-1 所示。



图 A-1　PyCharm Community 安装界面图(1)

(3) 程序默认安装位置为 C:\Program Files\JetBrains\PyCharm Community Edition 2018.2.5，单击 Browse 可选择自定义安装目录，本安装教程的自定义安装路径为 D:\DevTools\Pycharm，单击 Next 按钮，如图 A-2 所示。

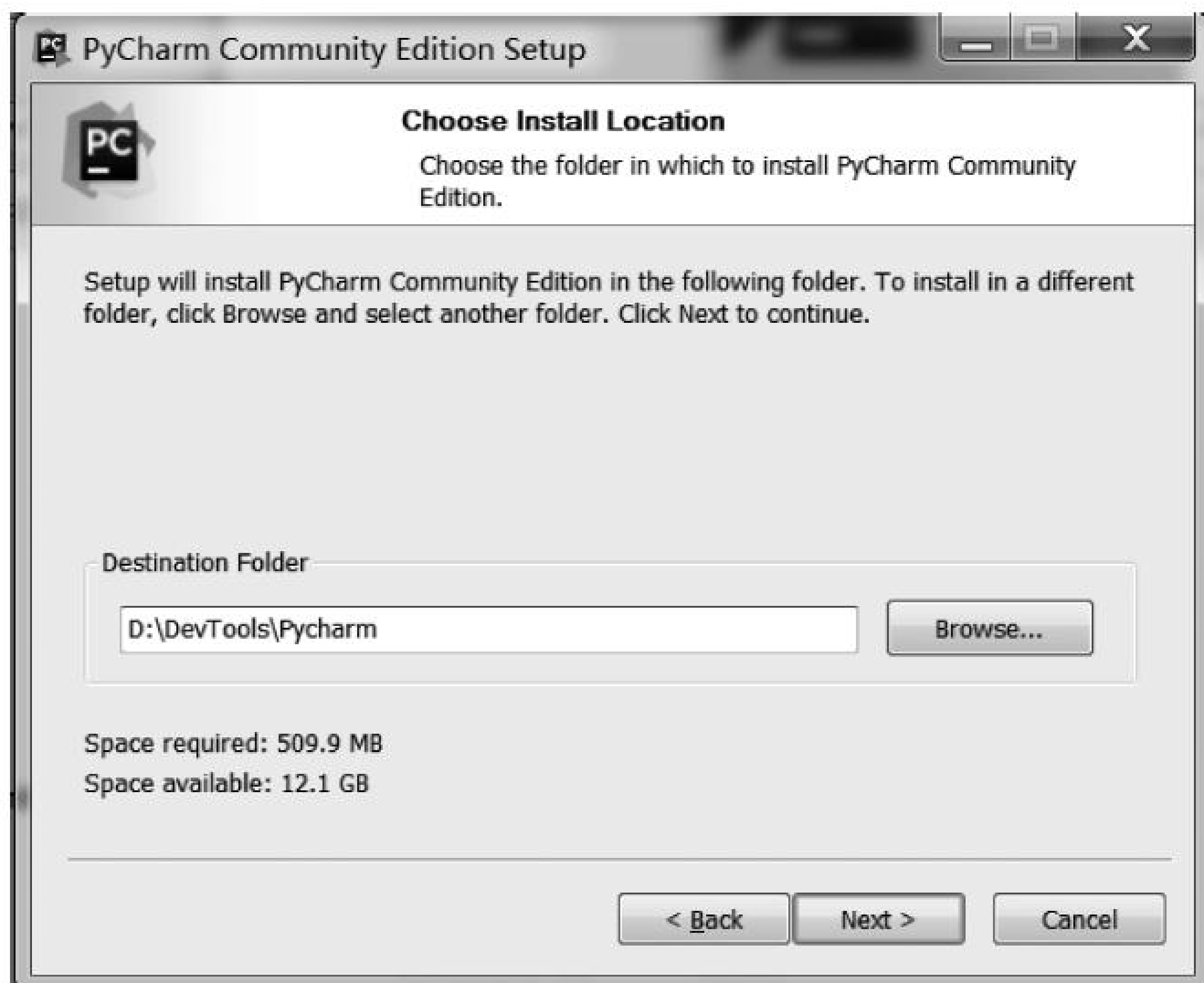


图 A-2　PyCharm Community 安装界面图(2)

(4) 勾选 64-bit launcher 和.py(如果是 32 位操作系统，则勾选 32-bit launcher)，单击 Next 按钮，如图 A-3 所示。

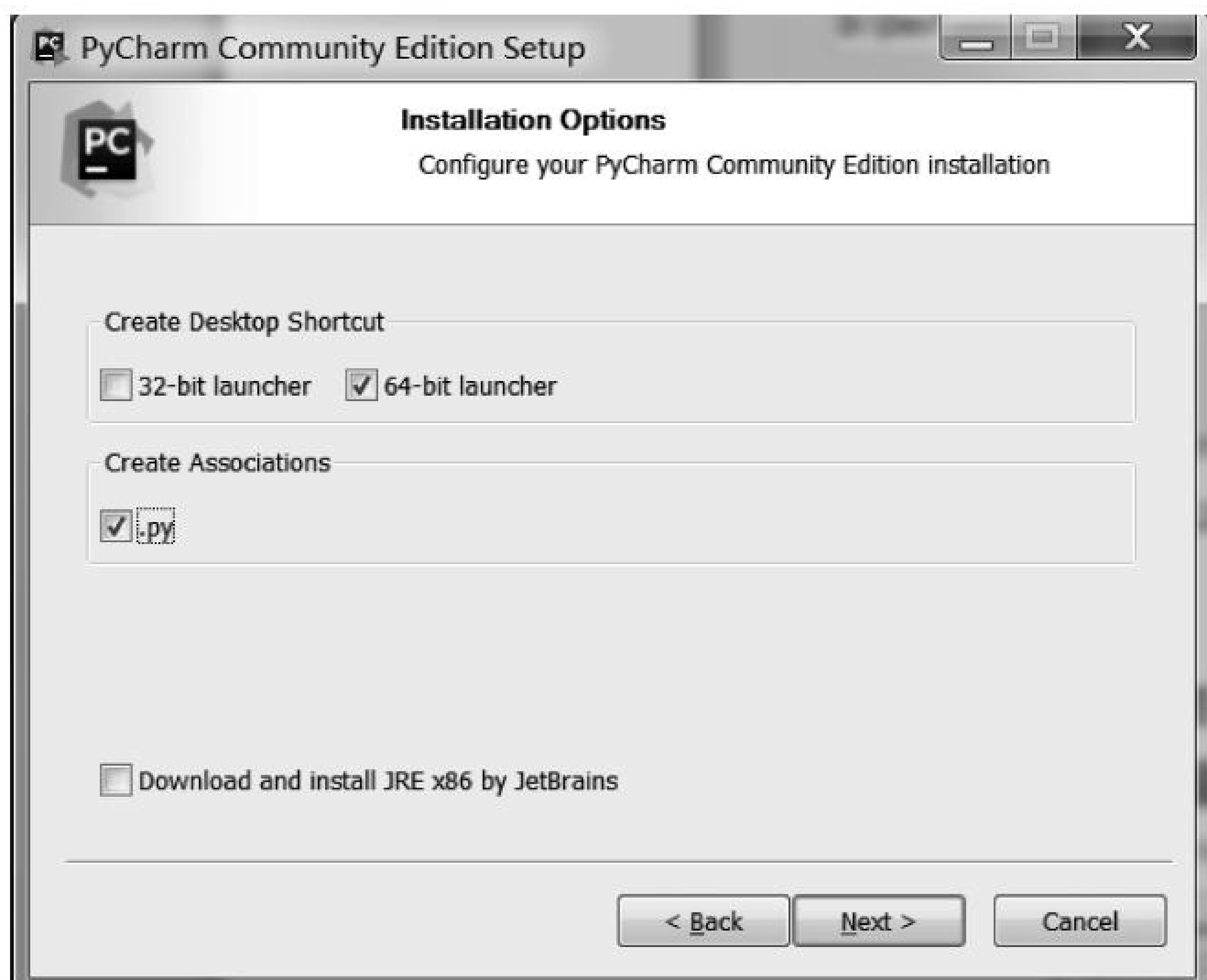


图 A-3　PyCharm Community 安装界面图(3)

(5) 单击 Install 按钮,如图 A-4 所示。



图 A-4 PyCharm Community 安装界面图(4)

(6) 安装大概需要 3～4 分钟,单击 Finish 按钮,至此完成了 PyCharm Community 的安装,如图 A-5 所示。

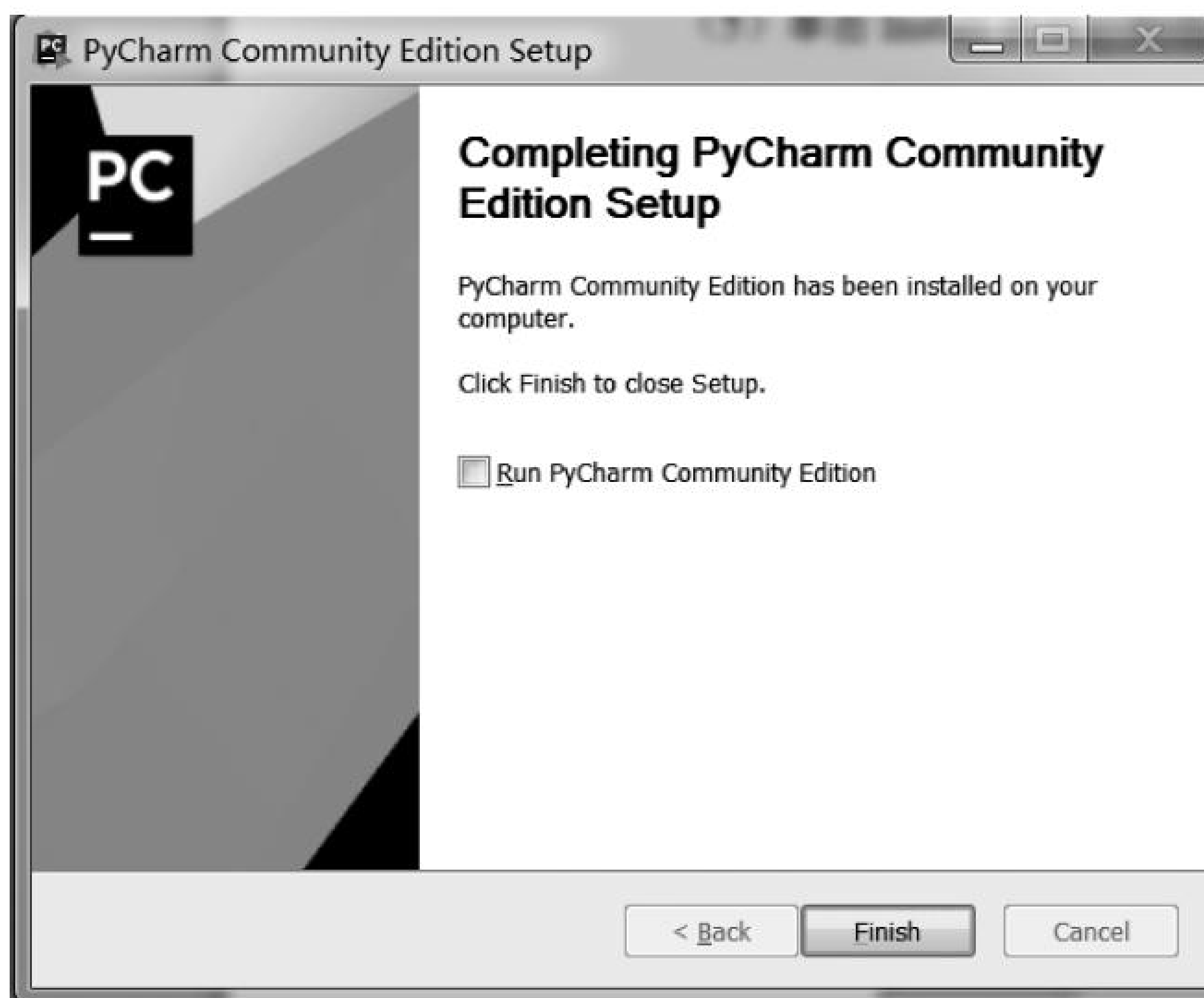


图 A-5 PyCharm Community 安装界面图(5)

附录B

MySQL安装步骤

MySQL 官网链接地址为：https://www. mysql. com/，下载链接地址为：https://www. mysql. com/downloads/。

本节以 MySQL 5. 7. 13 为例，进行安装说明。

(1) 右击“安装包”，单击“安装”按钮。

(2) 勾选 I accept the license terms，单击 Next 按钮，如图 B-1 所示。

(3) 默认勾选 Developer Default，本安装教程选择勾选 Custom，进行自定义安装，单击 Next 按钮，如图 B-2 所示。

(4) 本安装教程只选择安装 MySQL Server 5. 7. 13，单击 Next 按钮，如图 B-3 所示。

(5) 单击 Execute 按钮，开始安装 MySQL Server 5. 7. 13，如图 B-4 所示。

(6) 单击 Next 按钮，如图 B-5 所示。

(7) 单击 Next 按钮，如图 B-6 所示。

(8) 单击 Next 按钮，如图 B-7 所示。

(9) 输入两次密码，然后单击 Next 按钮，如图 B-8 所示。

(10) 单击 Next 按钮，如图 B-9 所示。

(11) 单击 Next 按钮，如图 B-10 所示。

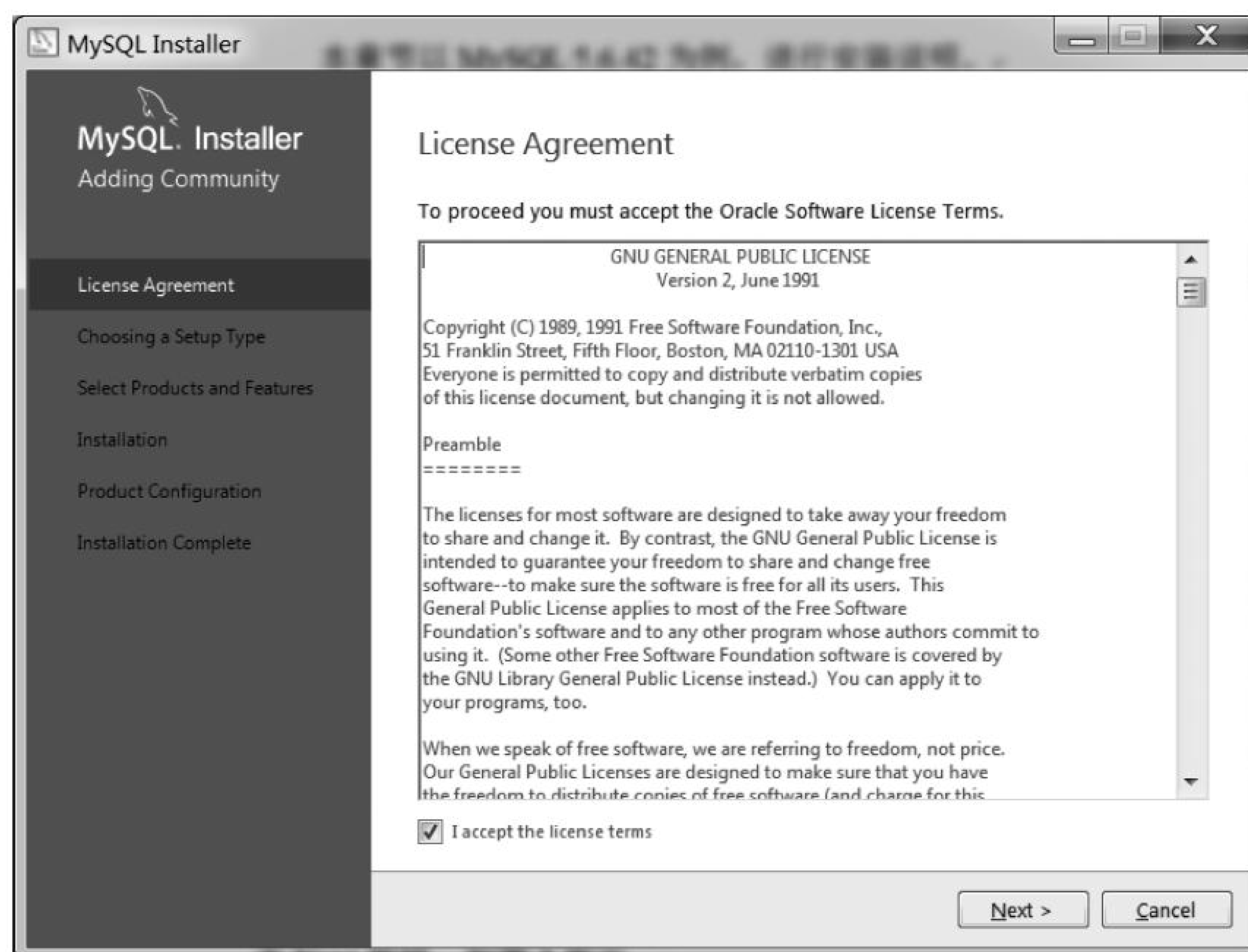


图 B-1　MySQL 5.7.13 安装界面图(1)

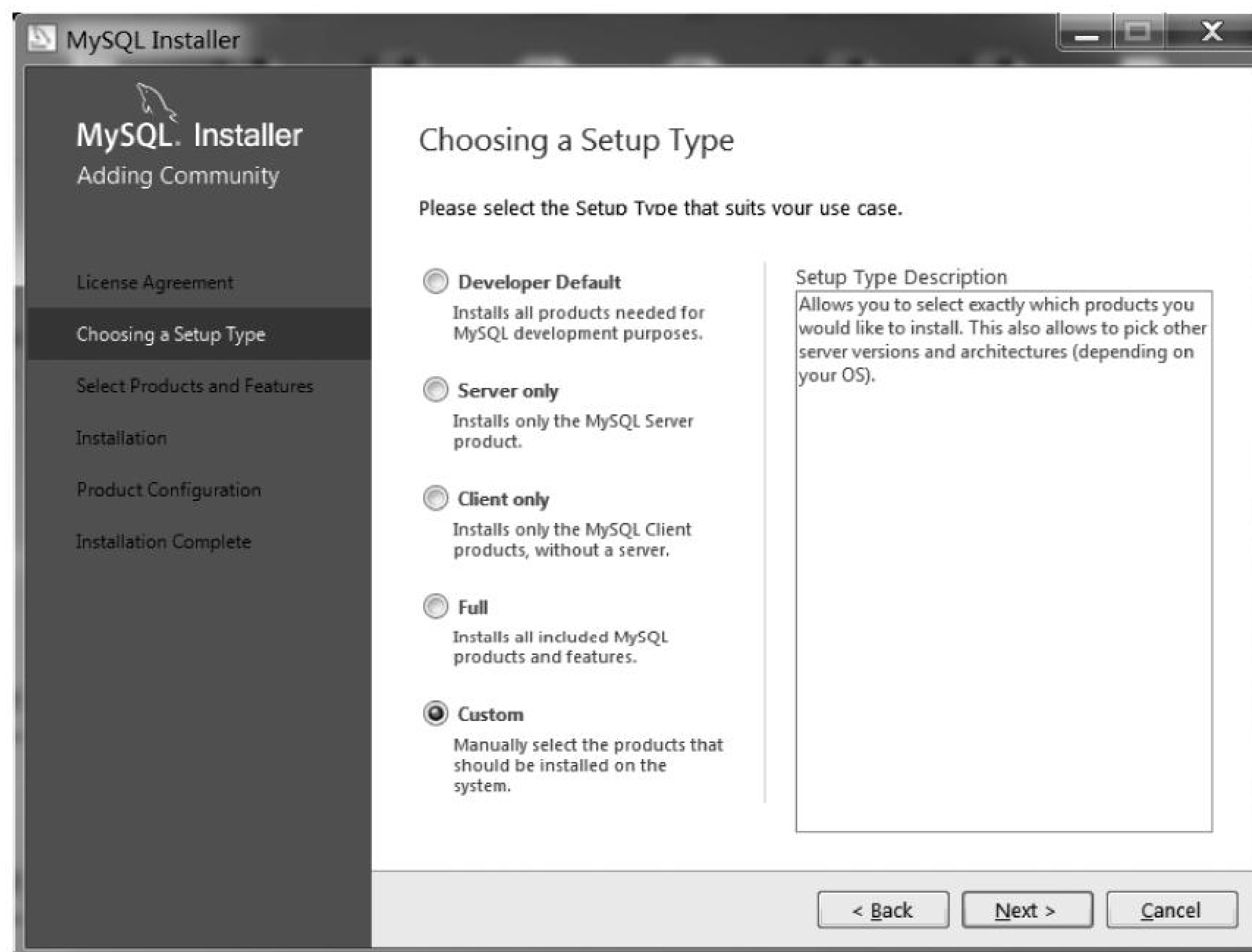


图 B-2　MySQL 5.7.13 安装界面图(2)

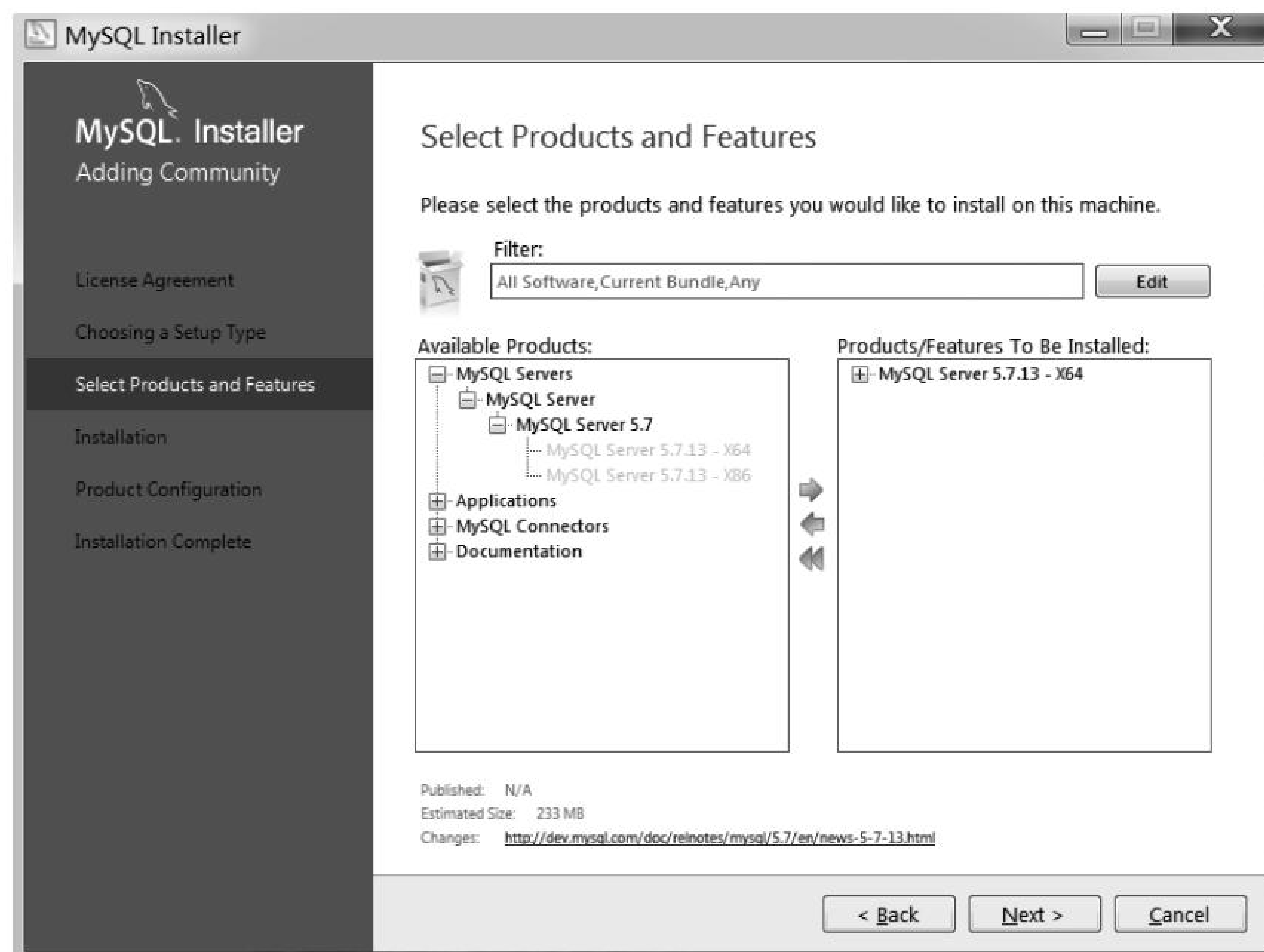


图 B-3　MySQL 5.7.13 安装界面图(3)

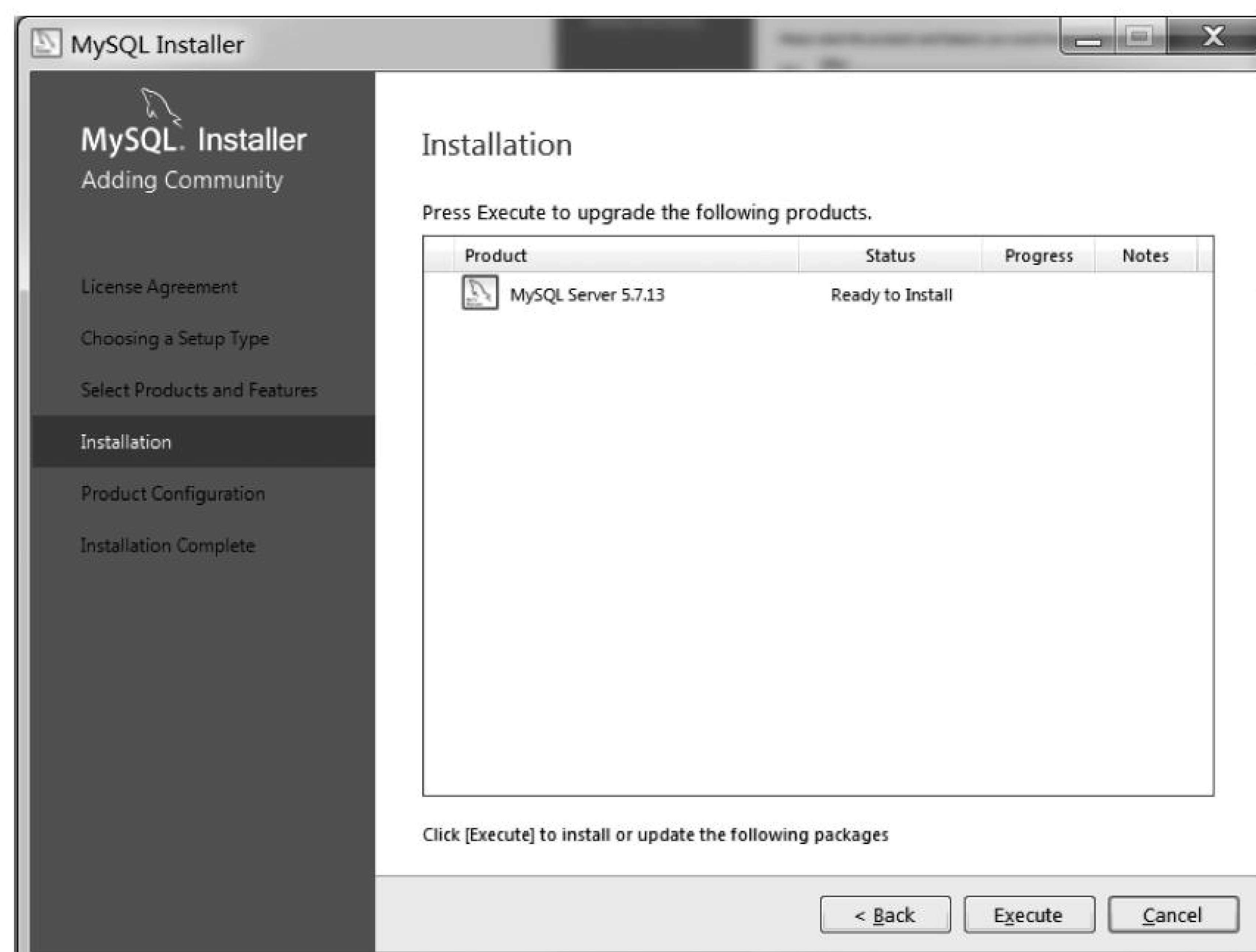


图 B-4　MySQL 5.7.13 安装界面图(4)

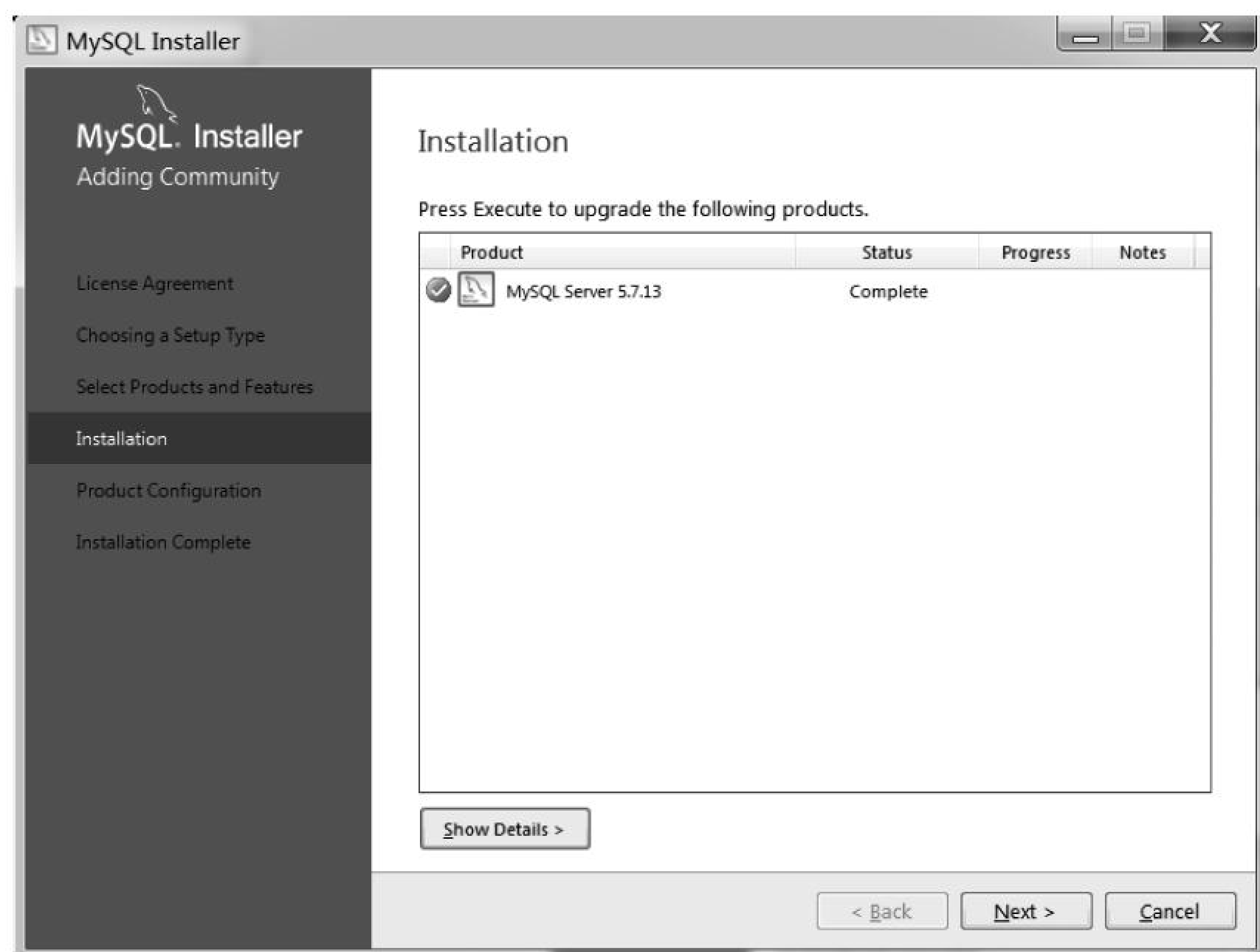


图 B-5　MySQL 5.7.13 安装界面图(5)

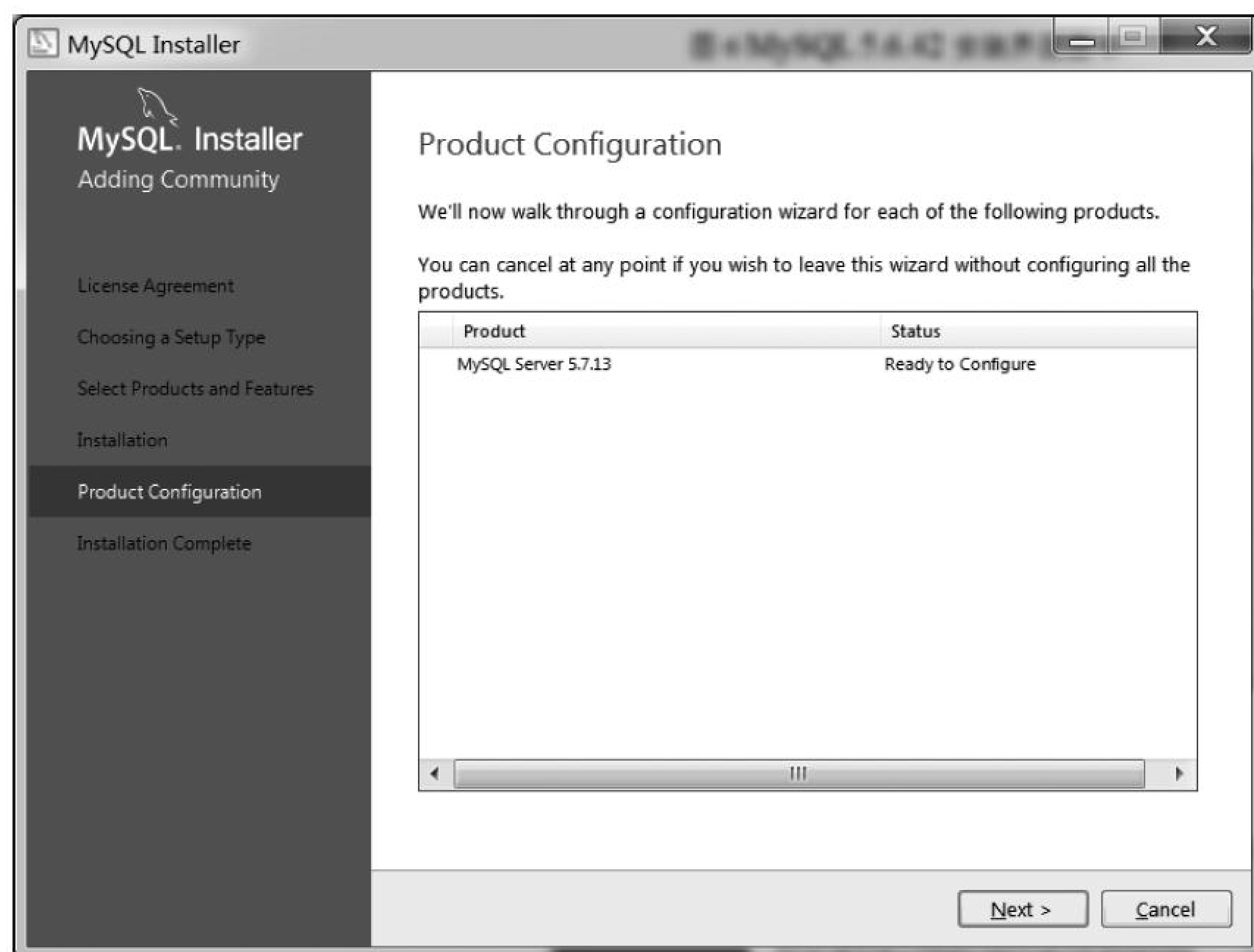


图 B-6　MySQL 5.7.13 安装界面图(6)

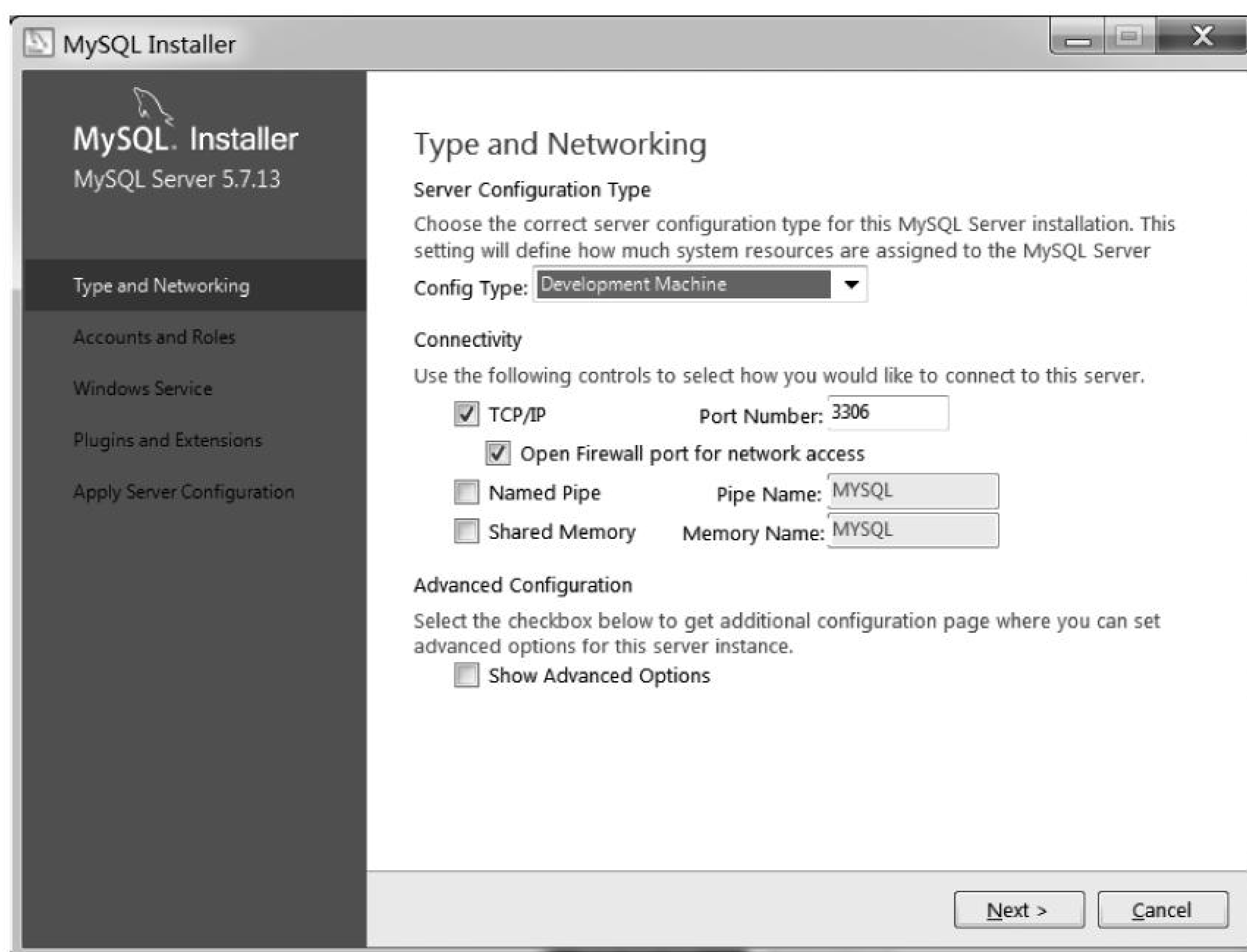


图 B-7　MySQL 5.7.13 安装界面图(7)

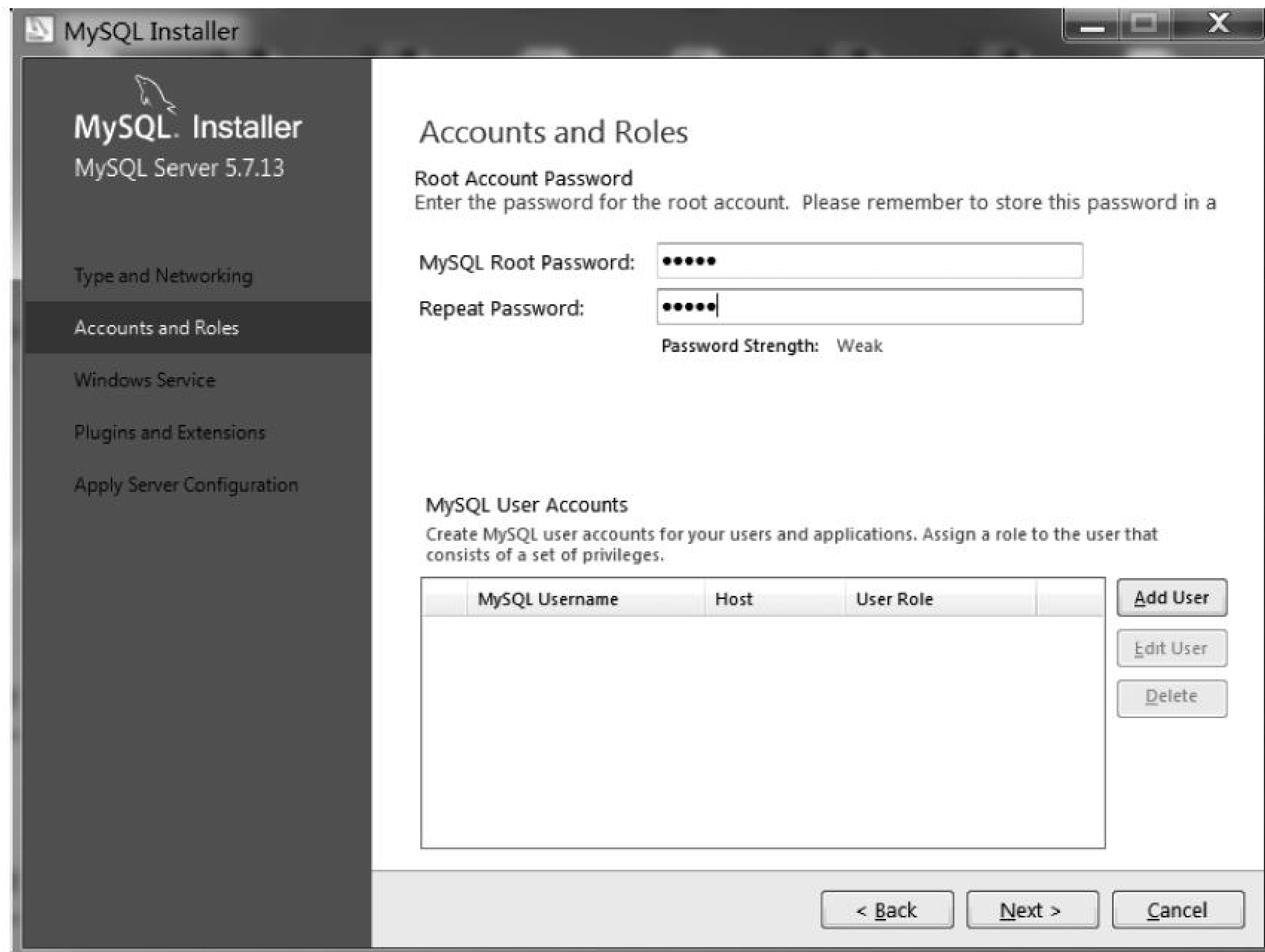


图 B-8　MySQL 5.7.13 安装界面图(8)

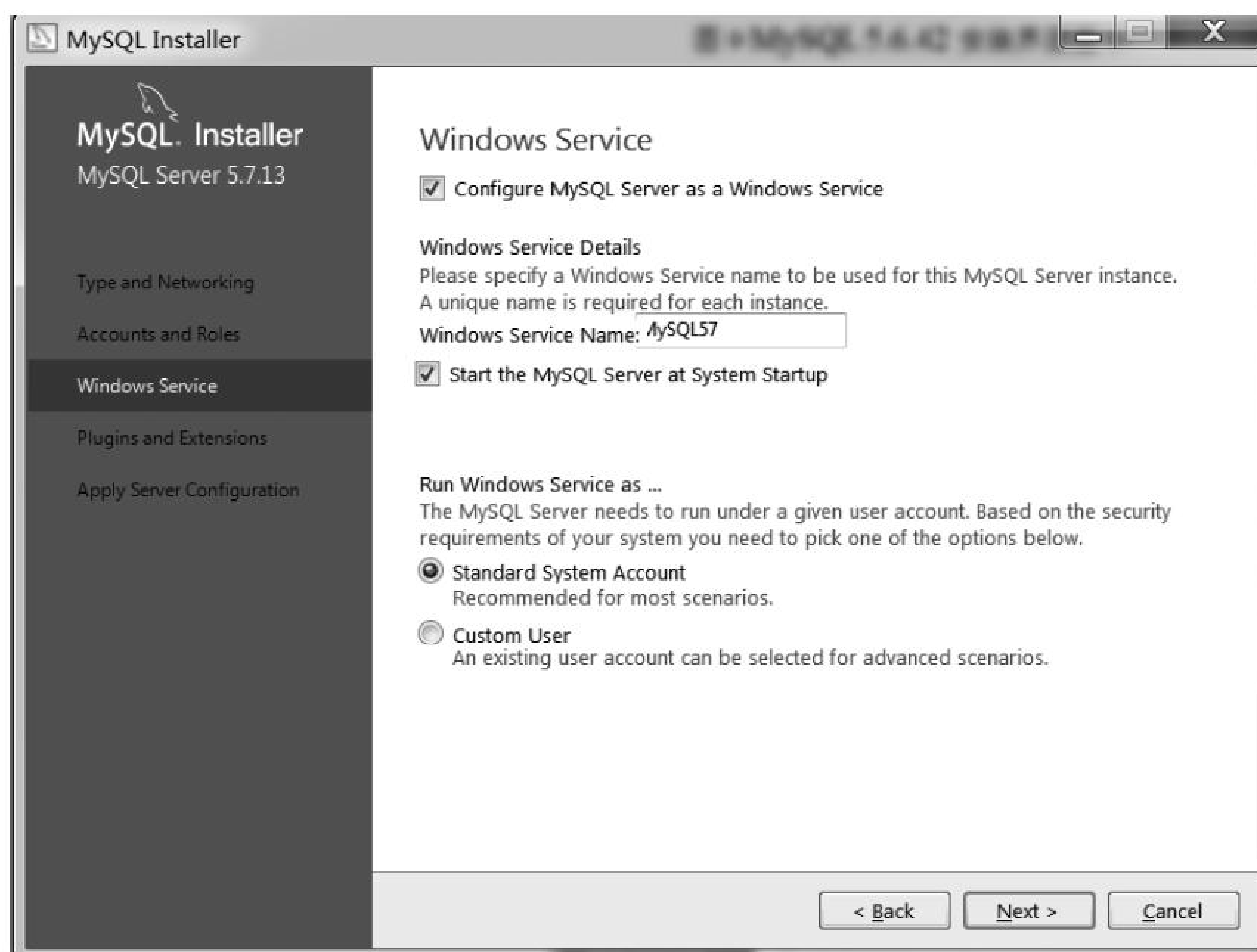


图 B-9 MySQL 5.7.13 安装界面图(9)

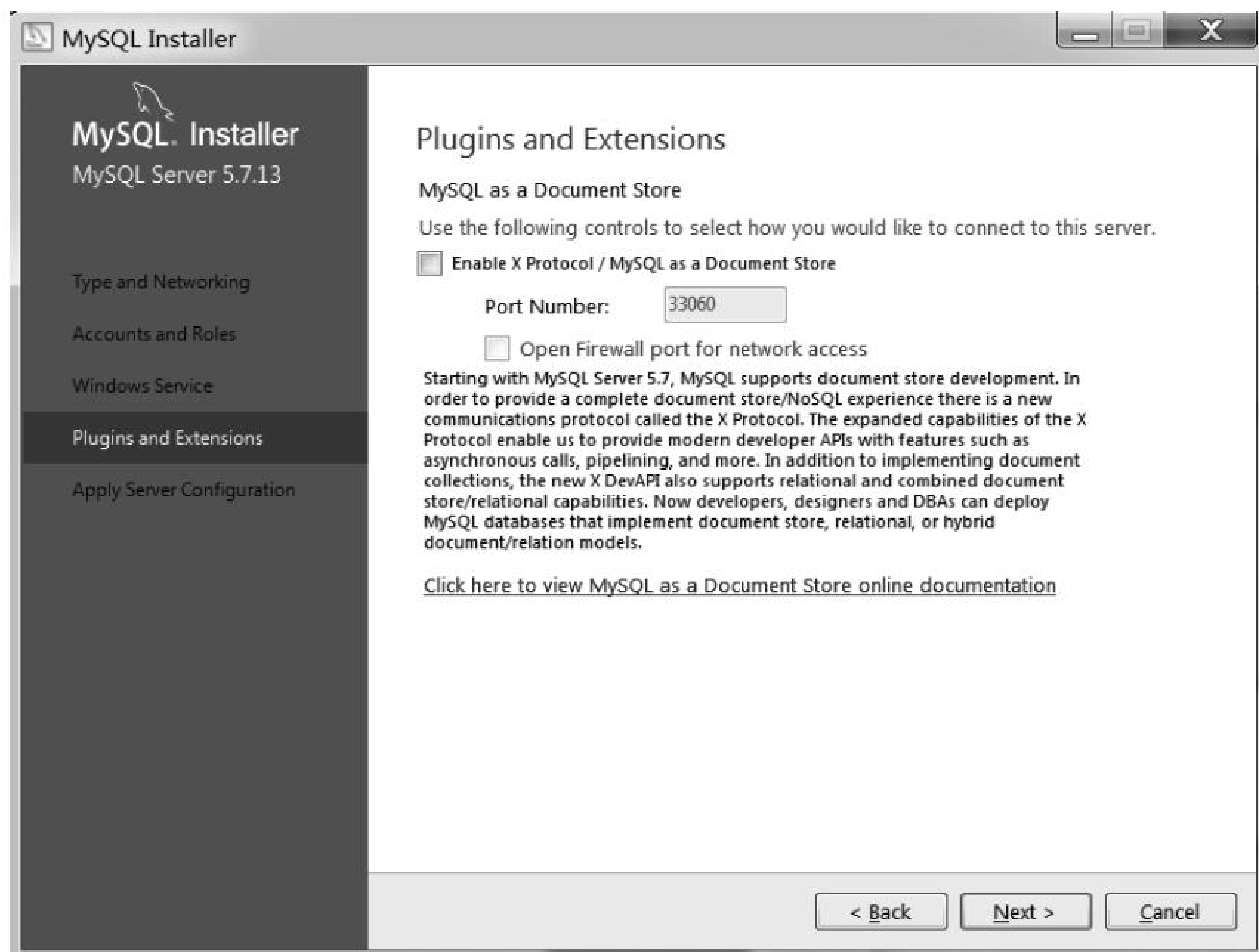


图 B-10 MySQL 5.7.13 安装界面图(10)

(12) 单击 Execute 按钮，如图 B-11 所示。

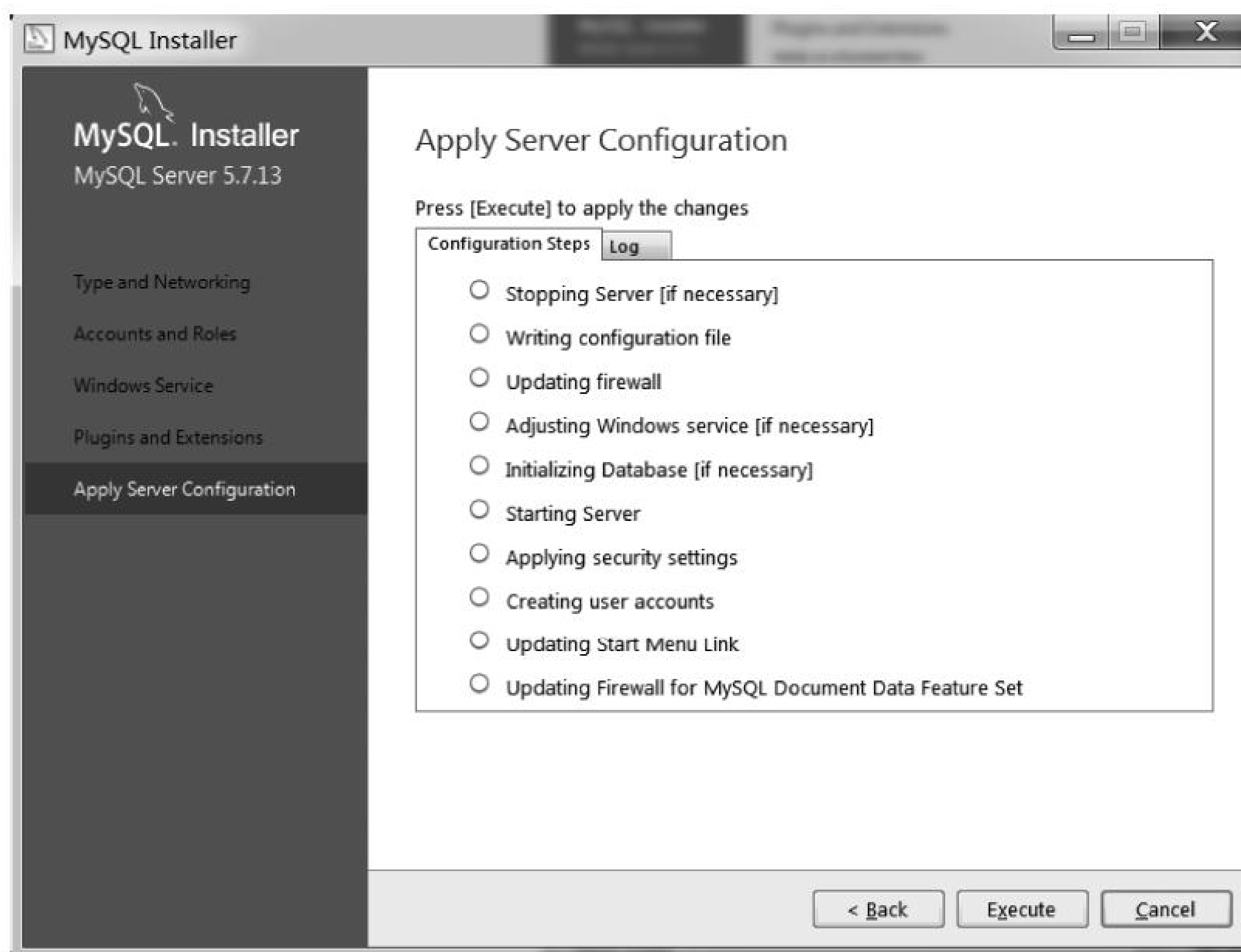


图 B-11　MySQL 5.7.13 安装界面图(11)

(13) 单击 Finish 按钮，完成 Product Configuration 的安装，如图 B-12 所示。

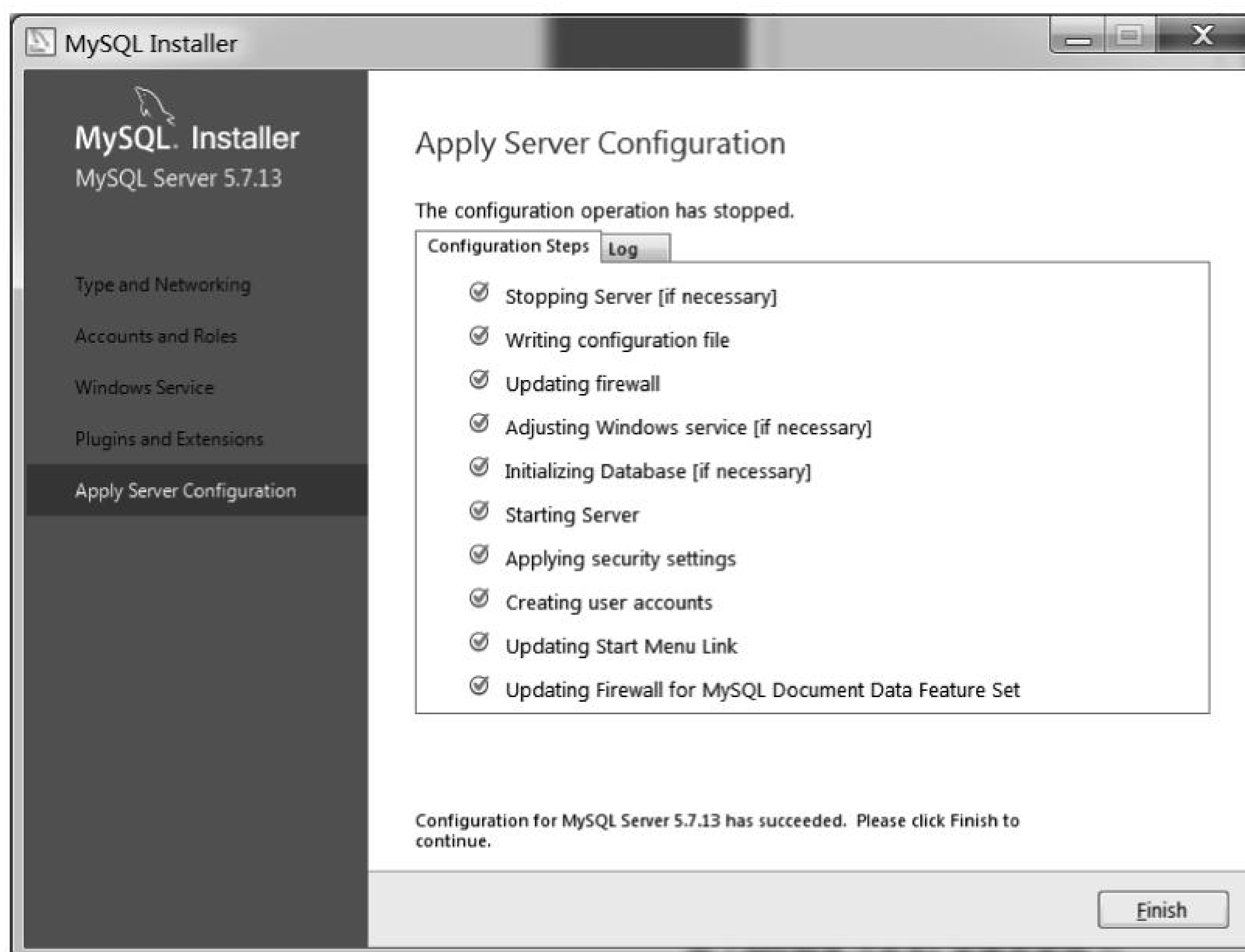


图 B-12　MySQL 5.7.13 安装界面图(12)

(14) 单击 Next 按钮，如图 B-13 所示。

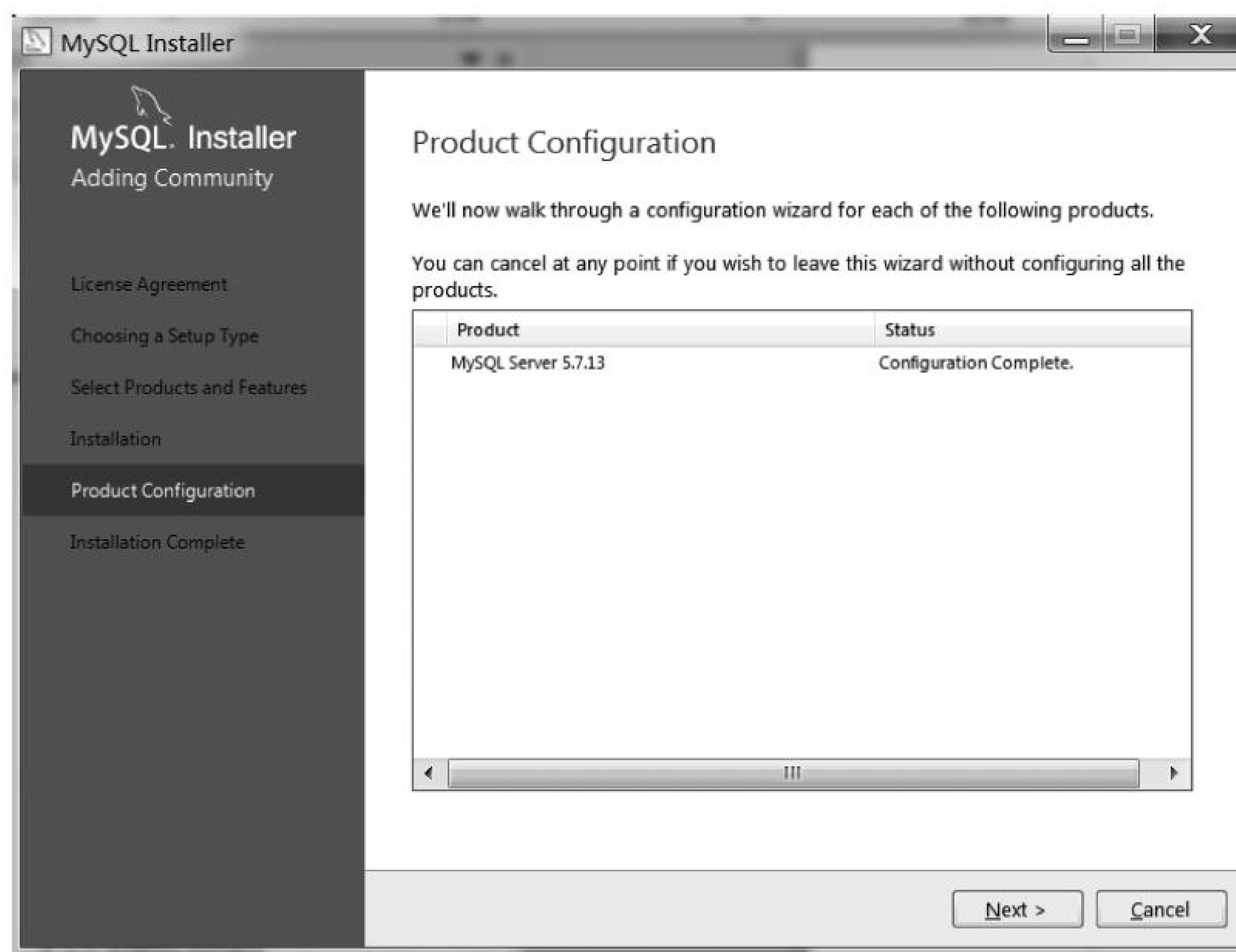


图 B-13 MySQL 5.7.13 安装界面图(13)

(15) 单击 Finish 按钮，至此完成了 MySQL 5.7.13 的安装，如图 B-14 所示。

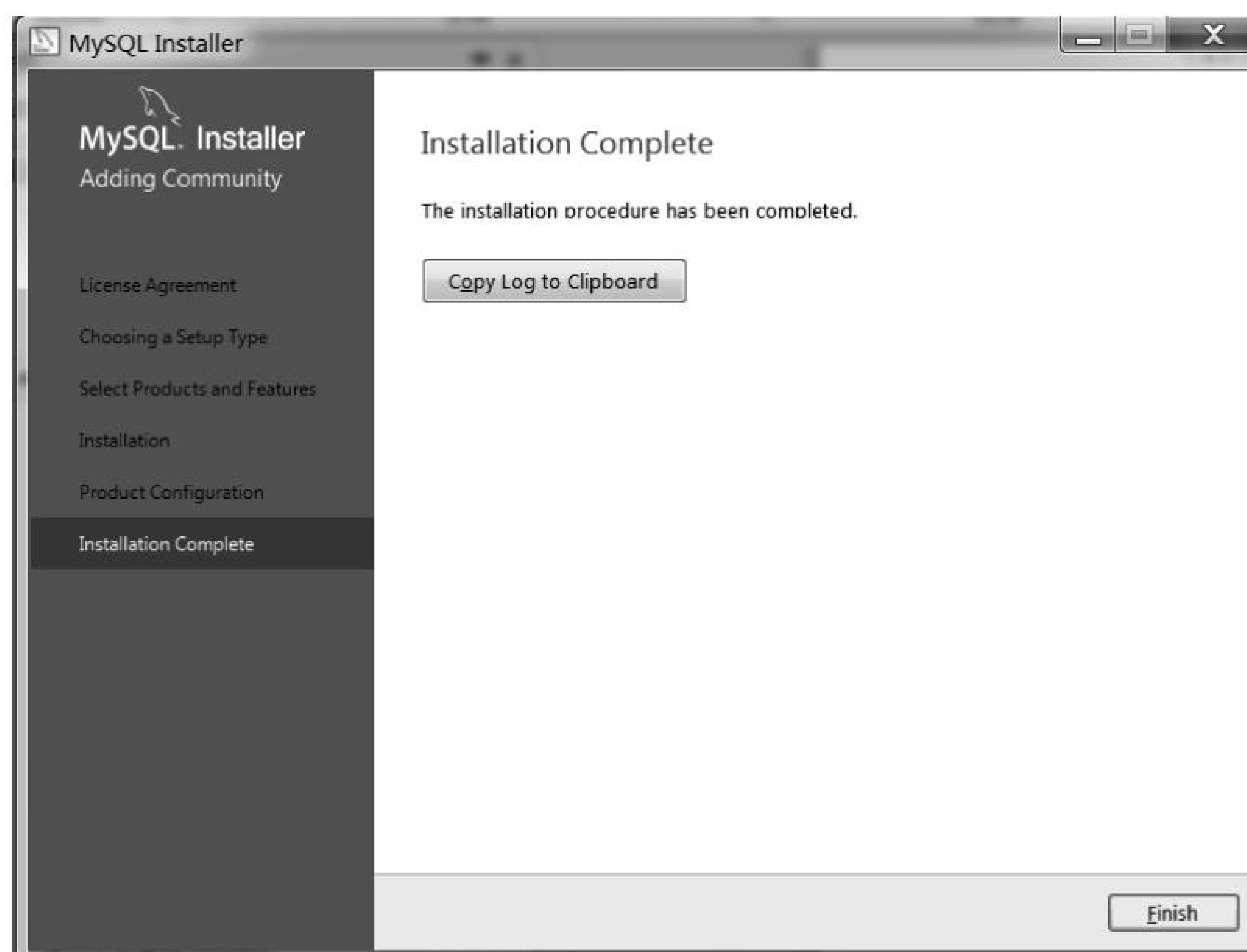


图 B-14 MySQL 5.7.13 安装界面图(14)

附录 C

MongoDB安装步骤

MongoDB 官网链接地址为：https://www.mongodb.com/，下载链接地址为：https://www.mongodb.com/download-center/community。

本节以 MongoDB 4.0.4 社区版为例，进行安装说明。

(1) 右击“安装包”，单击“安装”按钮。

(2) 单击 Next 按钮，如图 C-1 所示。

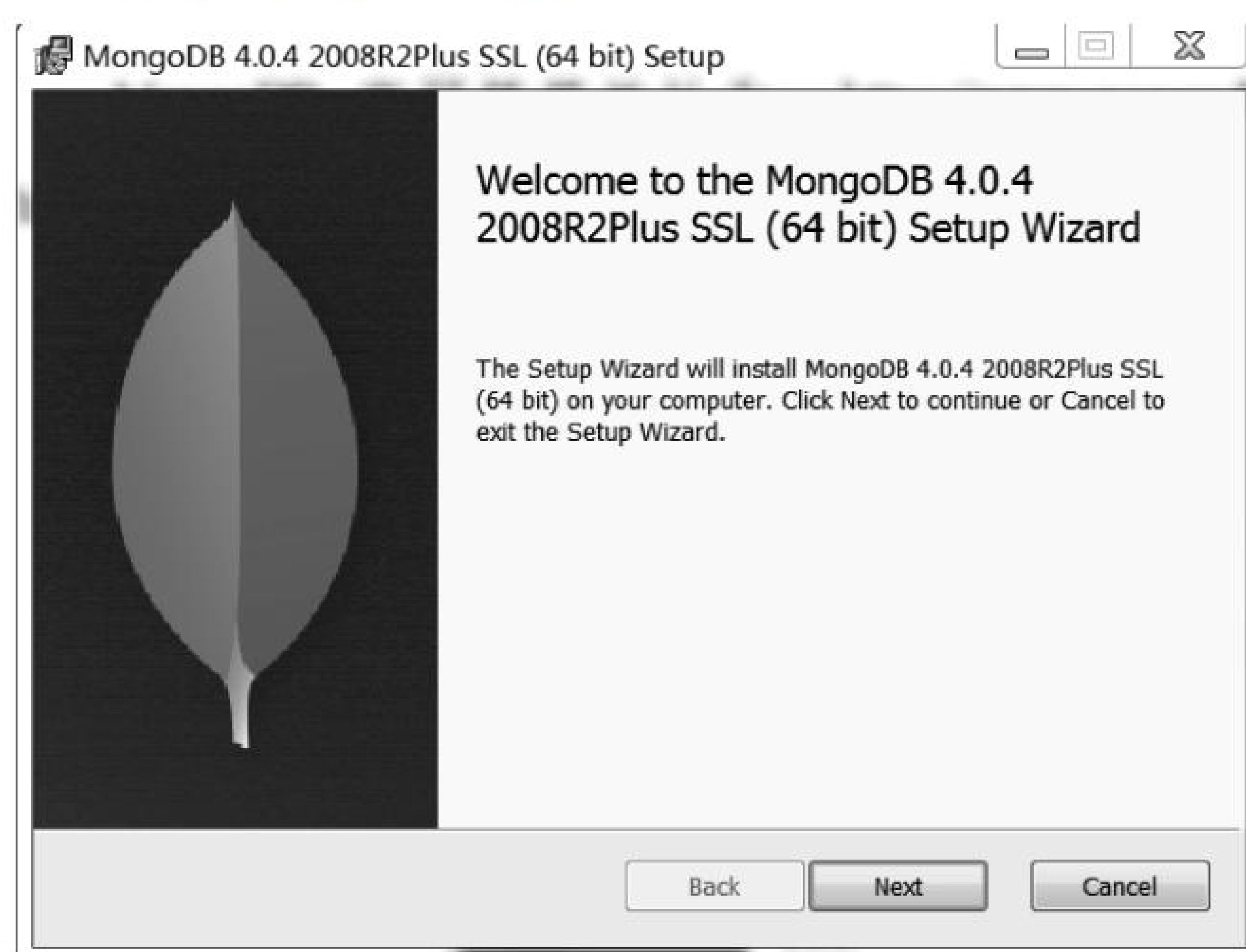


图 C-1　MongoDB 4.0.4 安装界面图(1)

(3) 勾选 I accept the terms in the License Agreement,单击 Next 按钮,如图 C-2 所示。

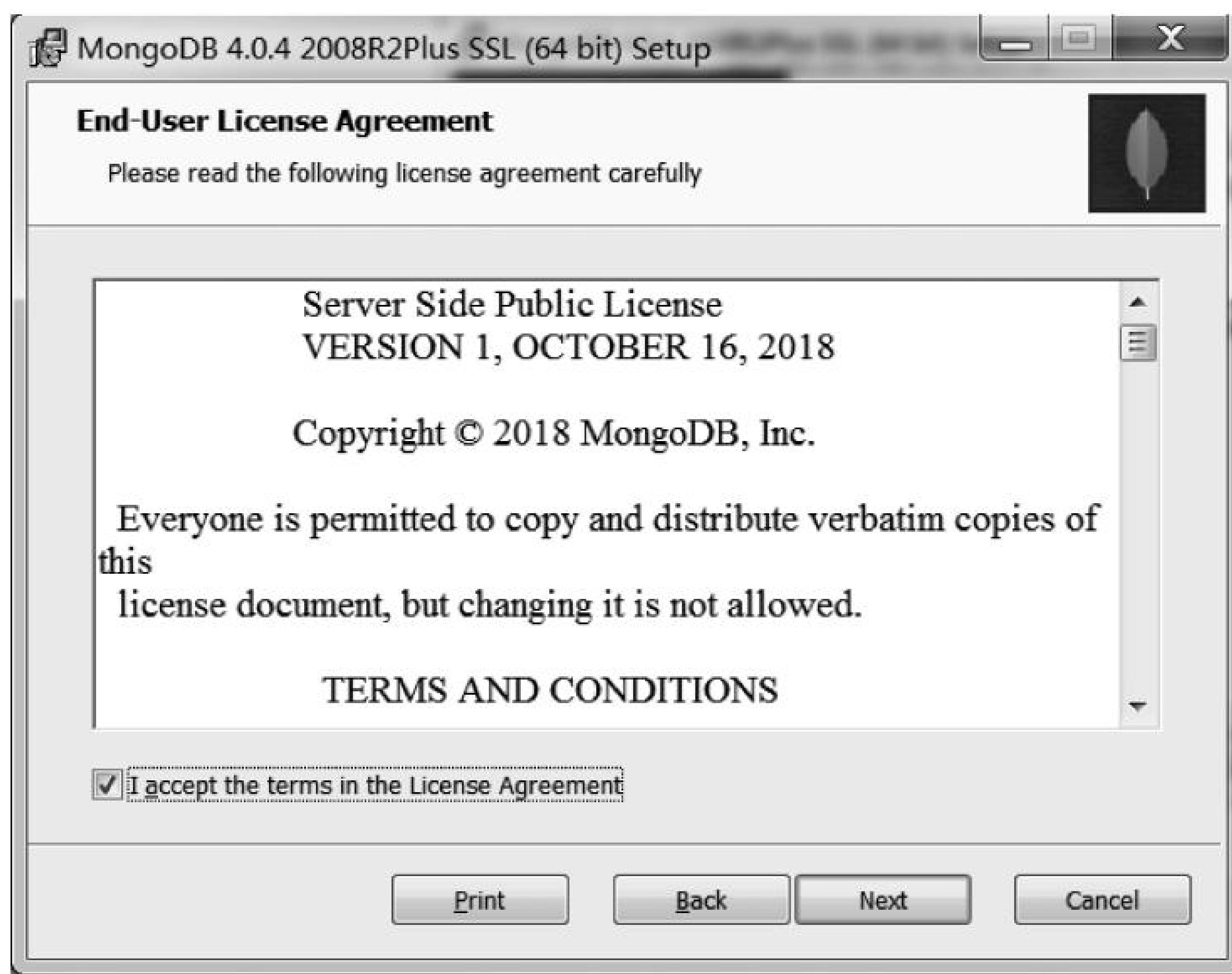


图 C-2 MongoDB 4.0.4 安装界面图(2)

(4) 本安装教程选择自定义安装,单击 Custom 按钮,如图 C-3 所示。

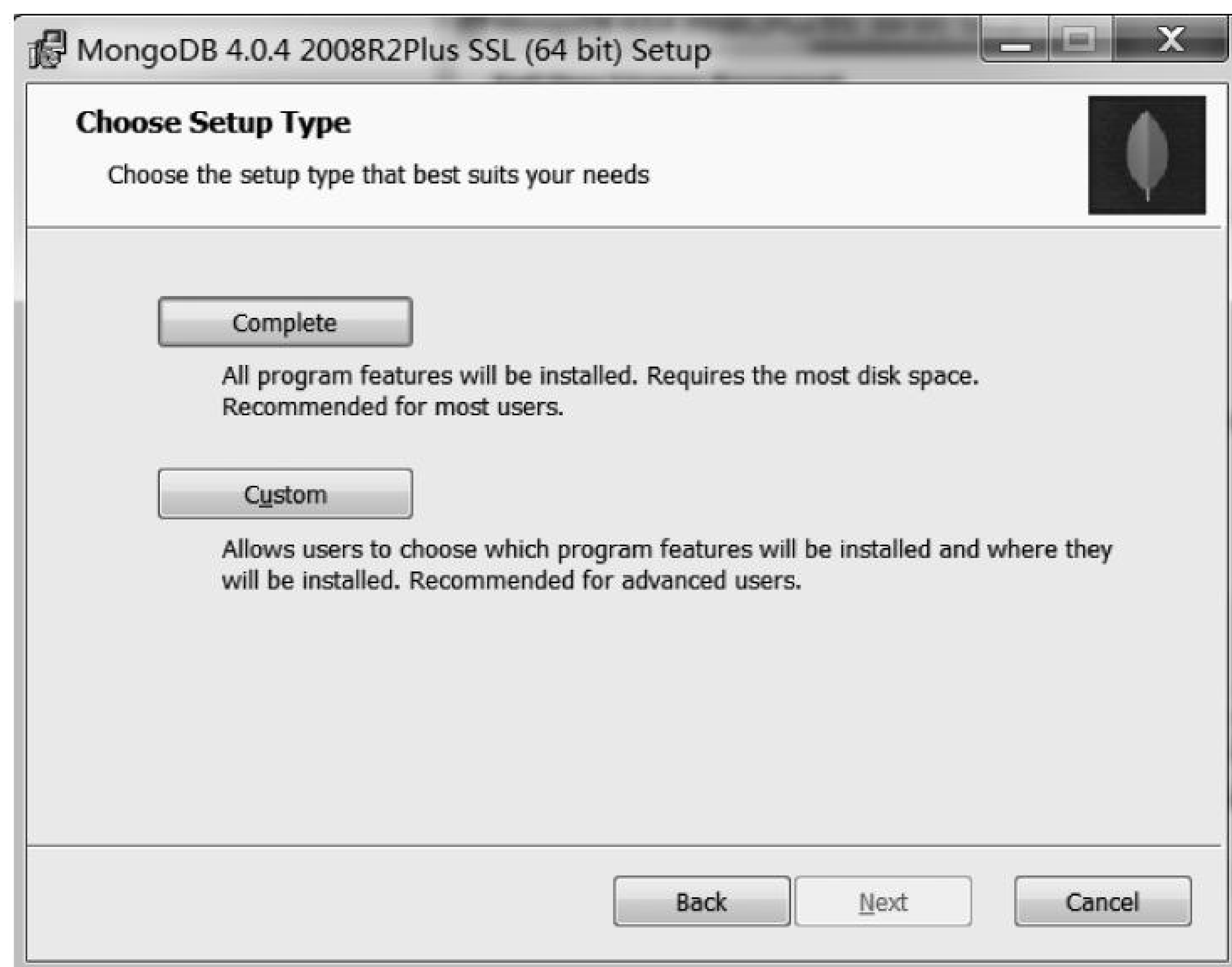


图 C-3 MongoDB 4.0.4 安装界面图(3)

(5) 程序默认安装位置为 C:\Program Files\MongoDB\Server\4.0\，单击 Browse 可选择自定义安装目录，本安装教程的自定义安装路径为 D:\DevTools\MongoDB\，单击 Next 按钮，如图 C-4 所示。

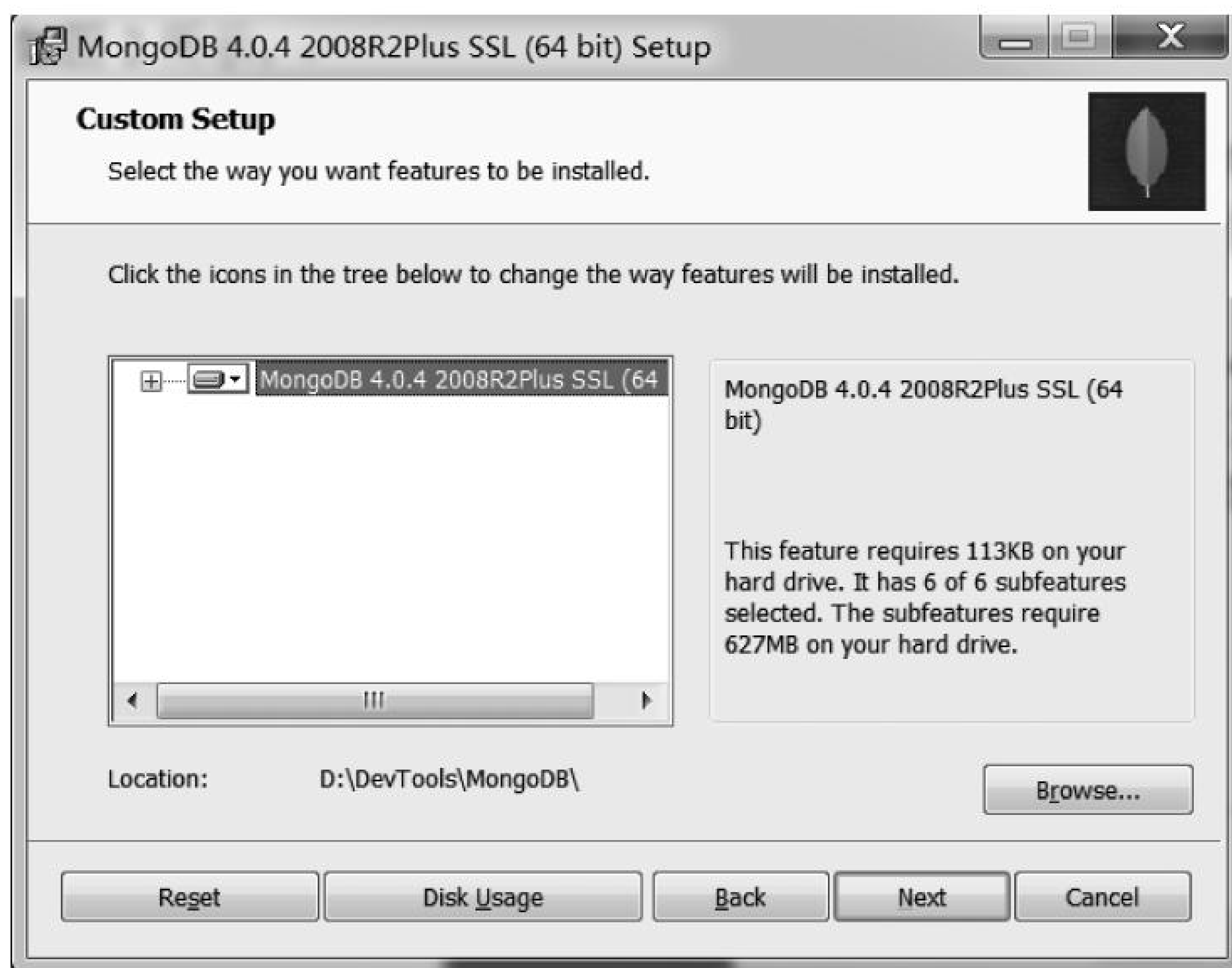


图 C-4　MongoDB 4.0.4 安装界面图(4)

(6) 单击 Next 按钮，如图 C-5 所示。

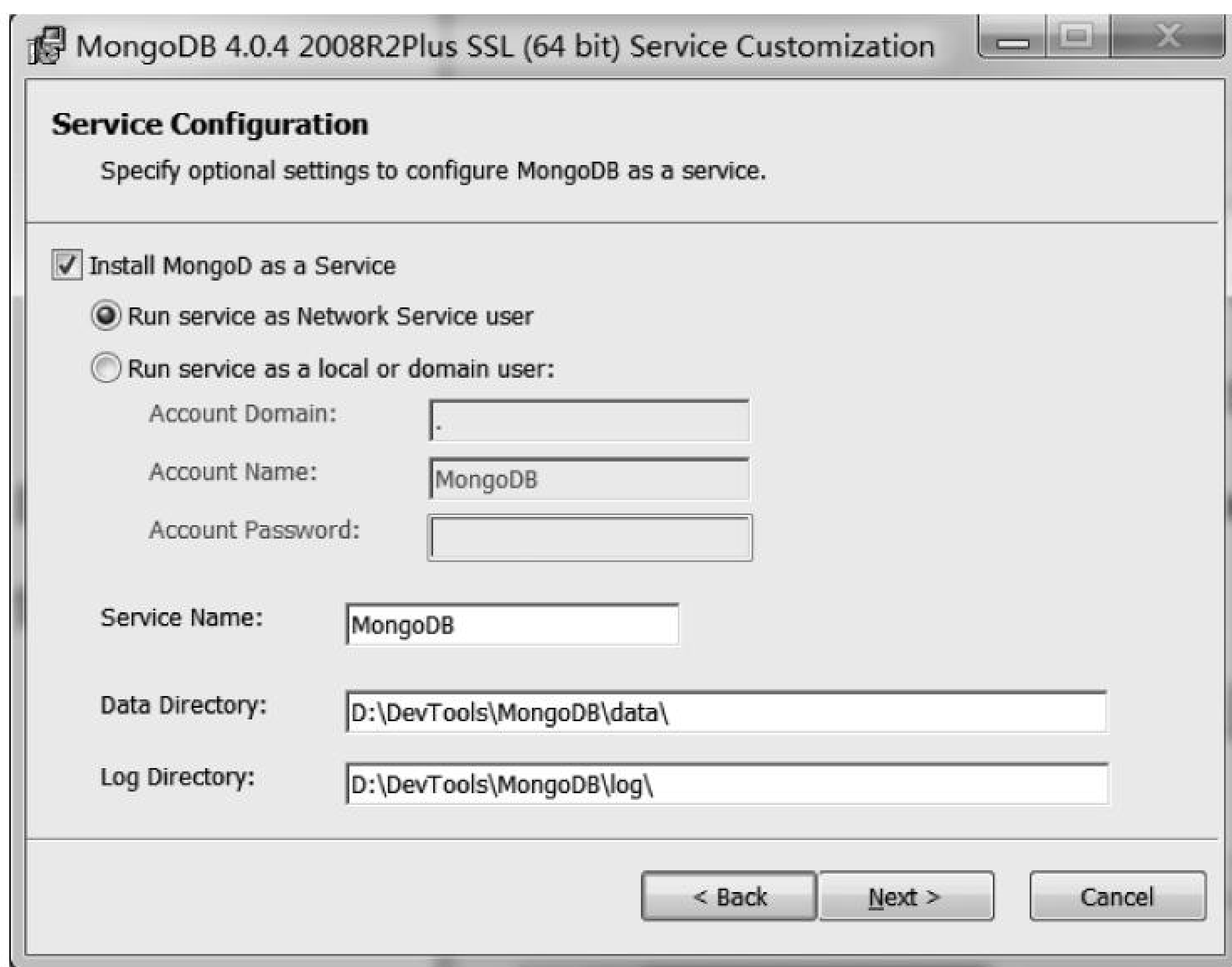


图 C-5　MongoDB 4.0.4 安装界面图(5)

(7) 不勾选 Install MongoDB Compass，单击 Next 按钮，如图 C-6 所示。

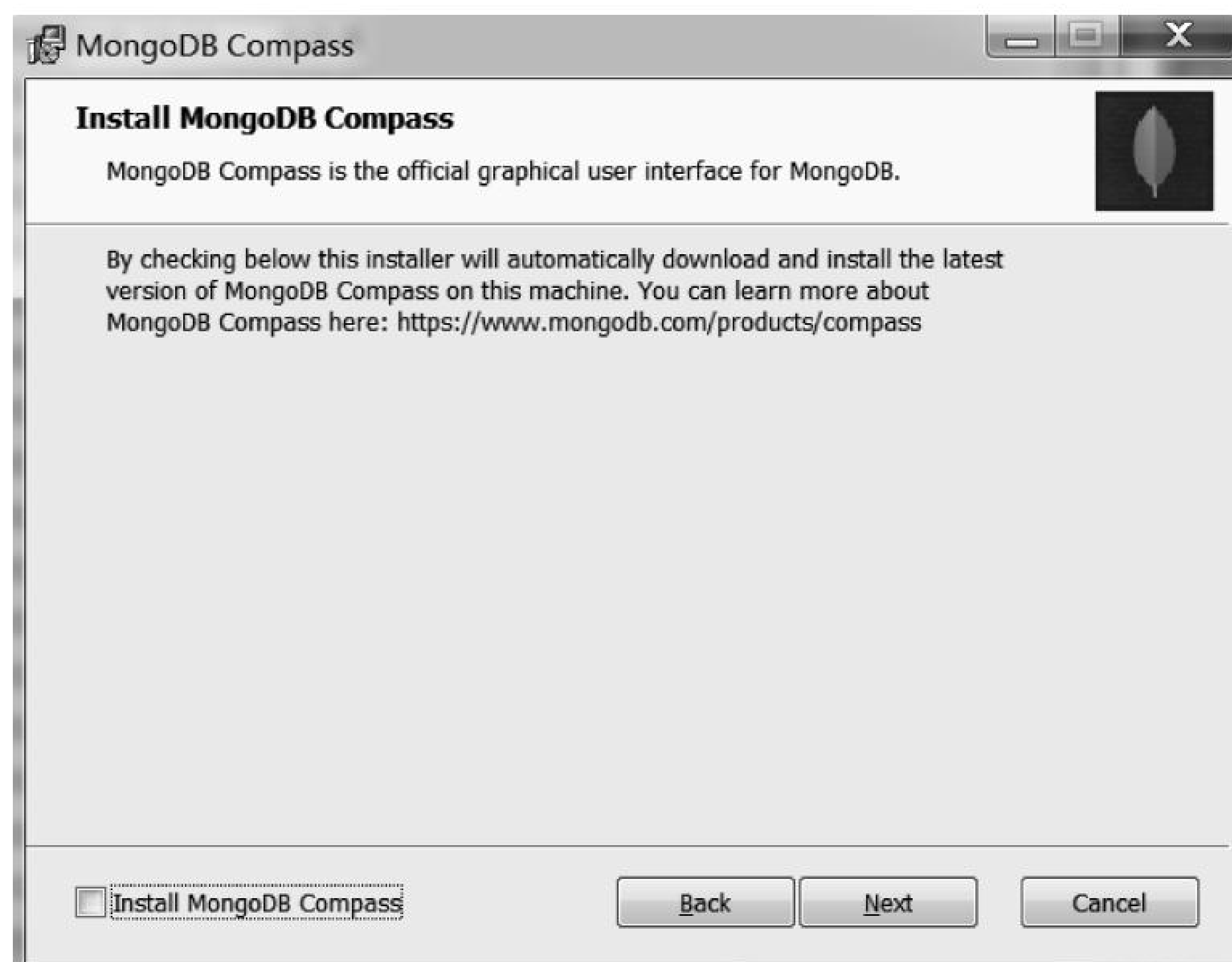


图 C-6 MongoDB 4.0.4 安装界面图(6)

(8) 单击 Install 按钮，如图 C-7 所示。

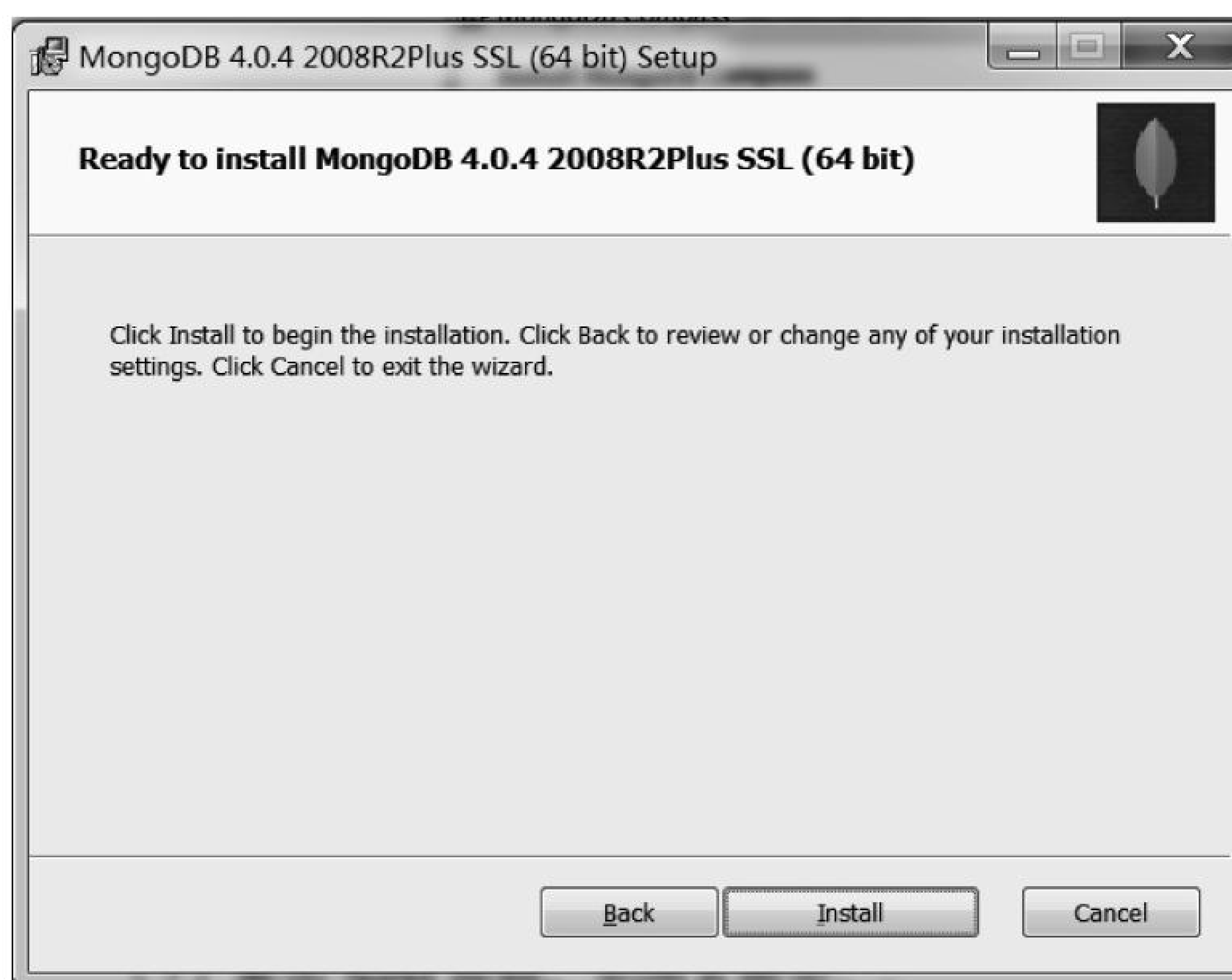


图 C-7 MongoDB 4.0.4 安装界面图(7)

（9）安装大概需要 3～4 分钟，单击 Finish 按钮，至此完成了 MongoDB 4.0.4 社区版的安装，如图 C-8 所示。

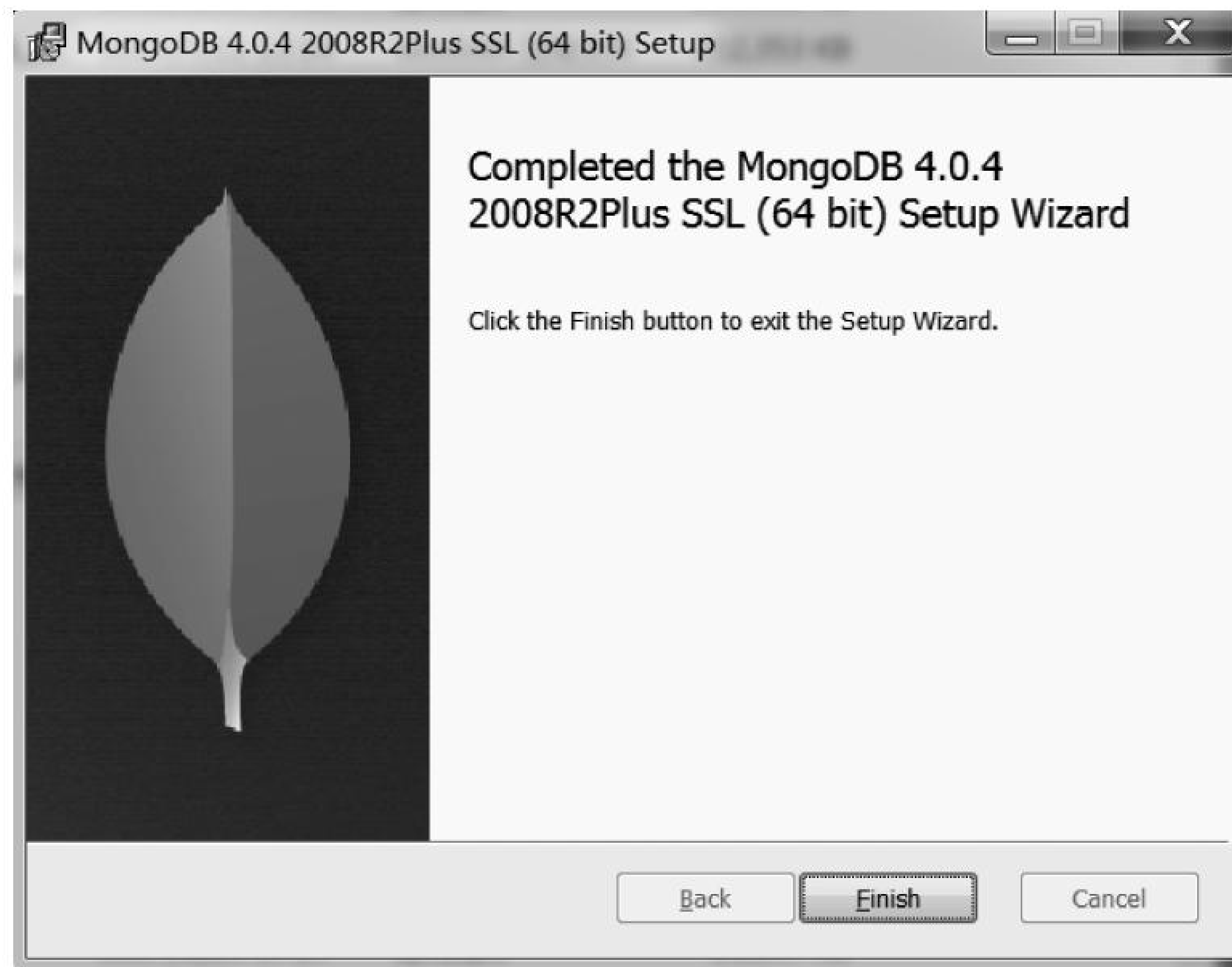


图 C-8　MongoDB 4.0.4 安装界面图(8)

附录 D

Neo4j安装步骤

Neo4j 官网链接地址为：https://neo4j.com/，下载链接地址为：https://neo4j.com/download-center/#releasesreleases/%23releases。

本节以 neo4j-3.4.9 社区版为例，进行安装说明。

（1）解压 zip 文件，将解压后的文件复制到自己想要的安装目录下，本安装教程的目录为：D:\DevTools\neo4j-community-3.4.9，如图 D-1 所示。

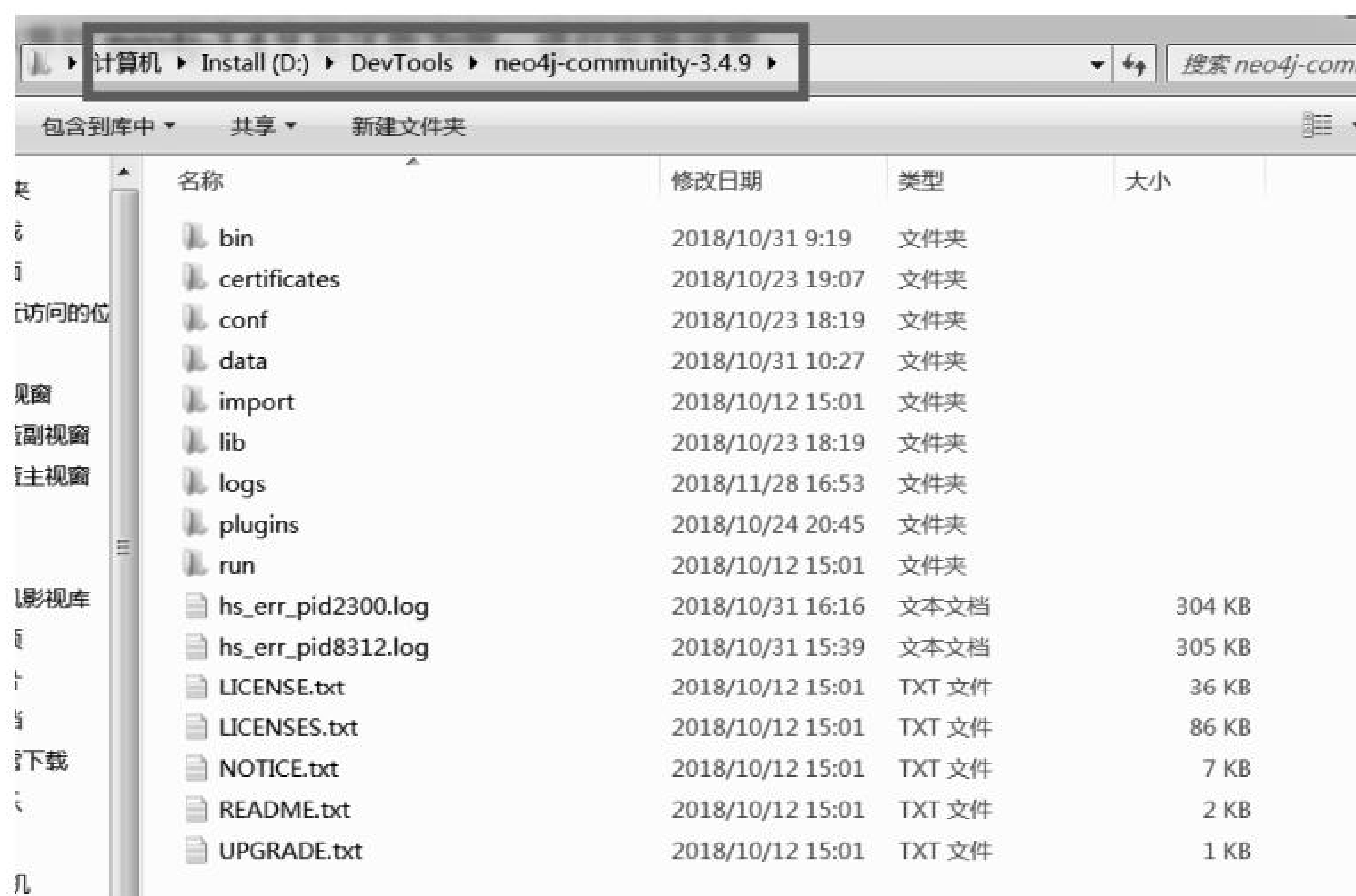


图 D-1　MongoDB 4.0.4 安装界面图

（2）Neo4j 环境配置，单击“计算机-属性-高级系统设置”，单击“环境变量”按钮。在“系统变量”栏下单击“新建”按钮，创建新的系统环境变量，如图 D-2 所示。



图 D-2　Neo4j 环境配置图(1)

（3）新建 NEO4J_HOME 变量，对应的变量值为 Neo4j 的安装目录，本教程安装目录为：D:\DevTools\neo4j-community-3.4.9，单击“确定”按钮，如图 D-3 所示。



图 D-3　Neo4j 环境配置图(2)

(4) 新建 PATH 变量,对应的变量值为 Neo4j 的安装目录的 bin,本教程安装目录为:D:\DevTools\neo4j-community-3.4.9\bin,单击“确定”按钮,如图 D-4 所示。



图 D-4 Neo4j 环境配置图(3)

(5) 检验 Neo4j 是否安装正确,可以打开 cmd 窗口,切换到安装目录的 bin 目录下:cd/d D:\DevTools\neo4j-community-3.4.9\bin,然后启动 Neo4j 服务。先后输入:neo4j install-service 和 neo4j start,即可启动 Neo4j 服务,如图 D-5 所示。

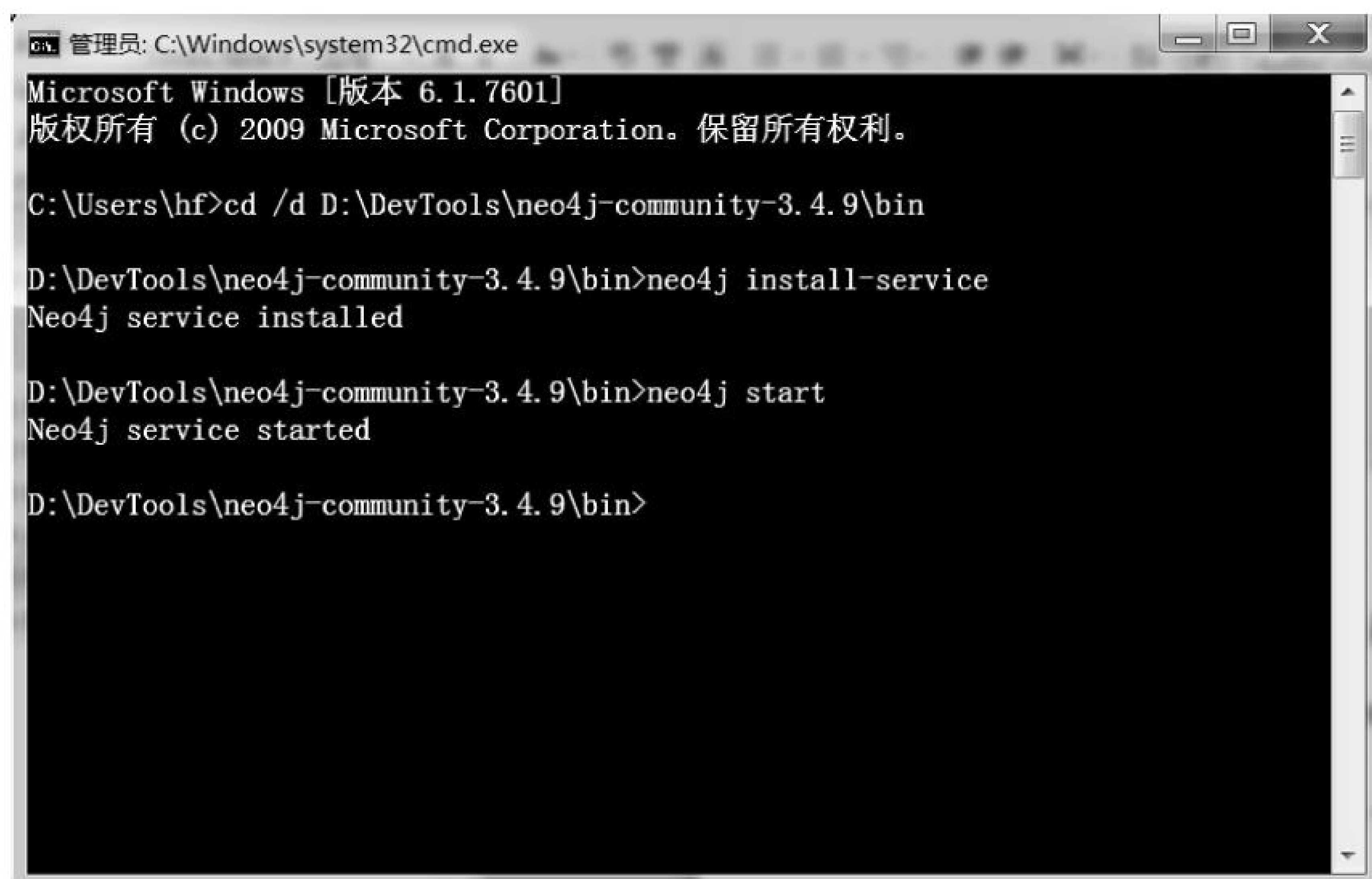


图 D-5 Neo4j 启动图

（6）打开浏览器，在浏览器中输入网址：http://localhost:7474/browser/，即可进入 Neo4j 界面，如图 D-6 所示。

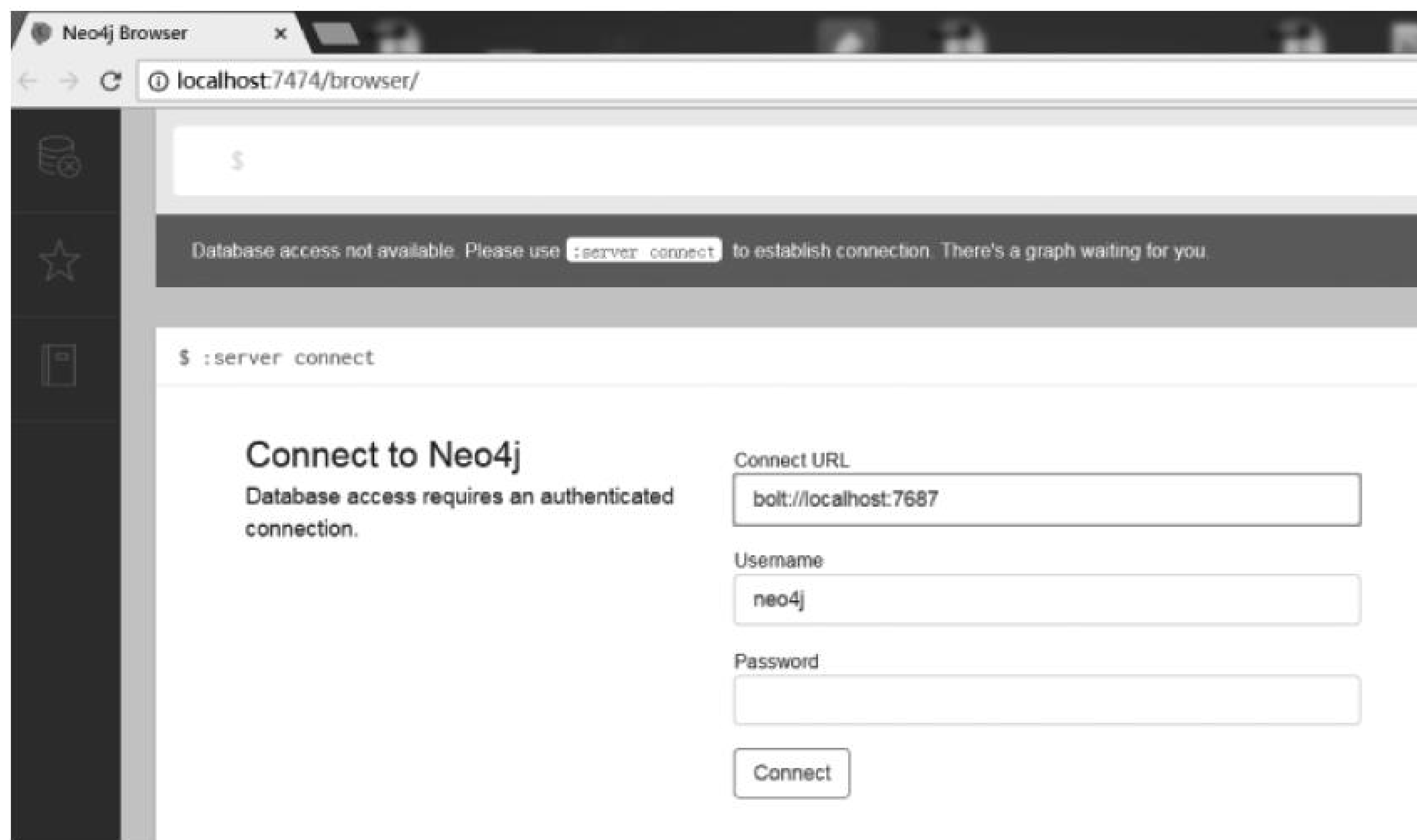


图 D-6　Neo4j 界面

附录 E

jdk安装步骤

jdk 下载链接地址为：https://www. oracle. com/technetwork/java/javase/downloads/index. html。

本节以 jdk-8u111 为例，进行安装说明。

（1）右击“安装包”，单击“以管理员身份运行”。

（2）单击“下一步”按钮，如图 E-1 所示。



图 E-1　jdk-8u111 安装界面图(1)

(3) jdk 默认安装位置为 C:\Program Files\Java\jdk1.8.0_111\，单击更改可选择自定义安装目录，本安装教程的自定义安装路径为 D:\DevTools\jdk1.8，单击“下一步”按钮，如图 E-2 所示。



图 E-2　jdk-8u111 安装界面图(2)

(4) jre 默认安装位置为 C:\Program Files\Java\jre1.8.0_111\，单击“更改可选择自定义安装目录”，本安装教程的自定义安装路径为 D:\DevTools\jre1.8，单击“下一步”按钮，如图 E-3 所示。



图 E-3　jdk-8u111 安装界面图(3)

（5）单击“关闭”按钮，至此完成了 jdk-8u111 的安装，如图 E-4 所示。



图 E-4 jdk-8u111 安装界面图(4)

（6）jdk 环境配置，单击“计算机-属性-高级系统设置”→“环境变量”。在“系统变量”栏下单击“新建”按钮，创建新的系统环境变量，如图 E-5 所示。



图 E-5 jdk 环境配置图(1)

（7）新建 JAVA_HOME 变量，对应的变量值为 jdk 的安装目录，本教程安装目录为：D:\DevTools\jdk1.8，单击“确定”按钮，如图 E-6 所示。



图 E-6　jdk 环境配置图(2)

(8) 新建 Path 变量，对应的变量值为 jdk 安装目录的 bin 和 jre\bin，本教程的变量值为：%JAVA_HOME%\bin；%JAVA_HOME%\jre\bin，单击“确定”按钮，如图 E-7 所示。



图 E-7　jdk 环境配置图(3)

(9) 新建 CLASSPATH 变量,取值为安装目录的 bin、lib\dt.jar 和 lib\tools.jar,本教程的变量值为:%JAVA_HOME%\lib;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar,单击"确定"按钮,如图 E-8 所示。



图 E-8 jdk 环境配置图(4)

(10) 检验 jdk 是否安装正确,可以打开 cmd 窗口,输入 java-version,如果能够查看到 Java 版本信息,说明安装正确,如图 E-9 所示。



图 E-9 java-version 命令图

附录F

第三方包安装步骤

本节以第三方包 imblearn 为例，进行安装说明。

(1) 打开 Anaconda 文件夹下的 Anaconda Prompt 窗口，如图 F-1 所示。

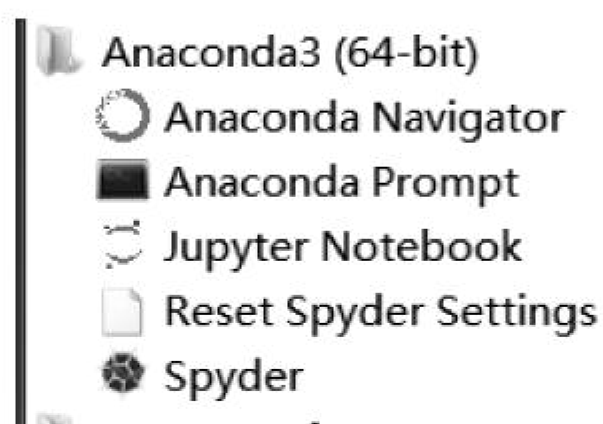


图 F-1　Anaconda Prompt 界面图

(2) 在窗口输入：pip install imblearn，如图 F-2 所示。

(3) 等待安装，提示 Successfully installed imbalanced-learn-0.4.3，即为安装成功，如图 F-3 所示。

对应的第三方包的安装，都是打开 Anaconda Prompt 命令窗口，输入：pip install 第三方包名。例如，pip install jiebe，pip install wordcloud 等。

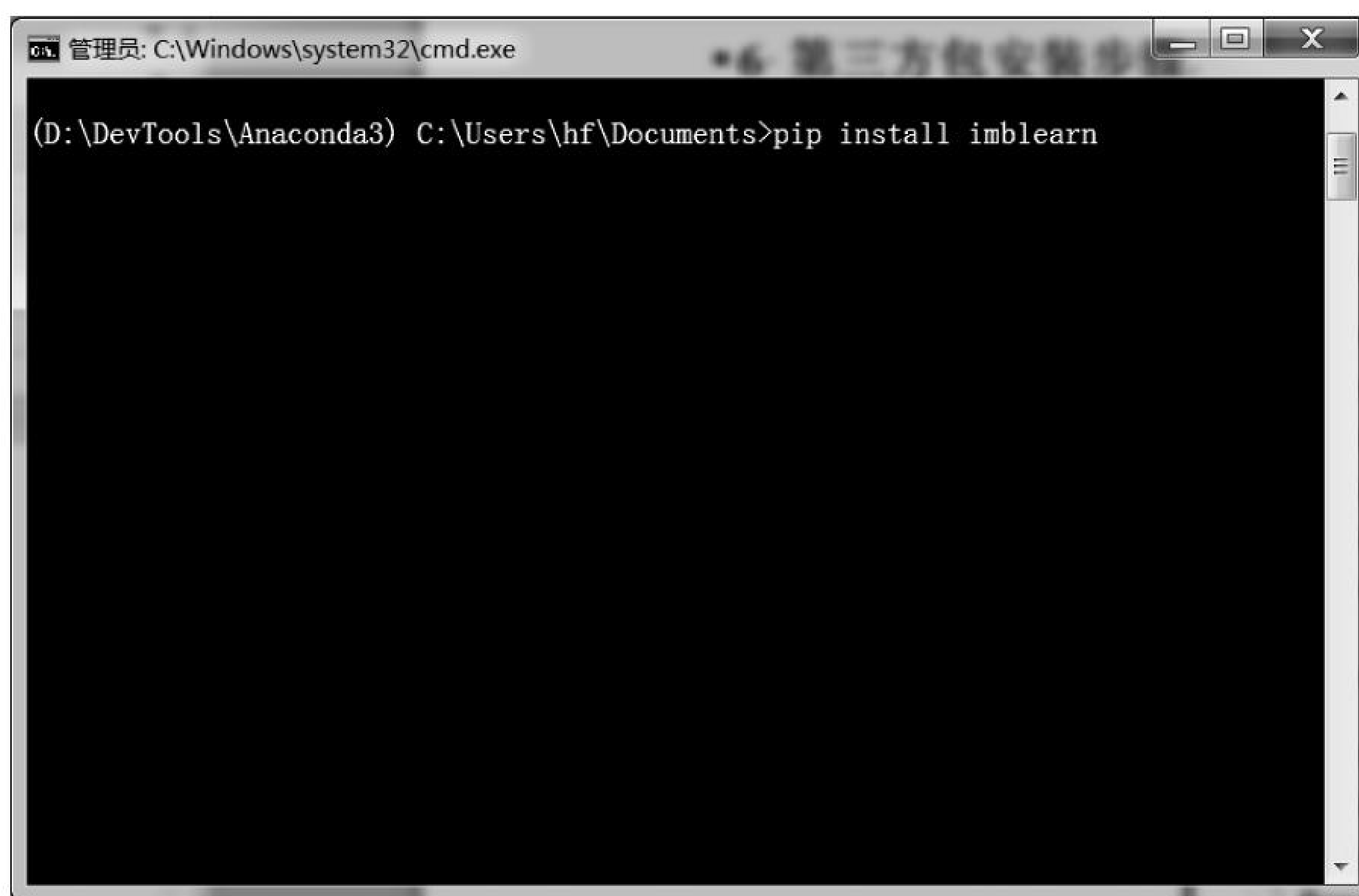


图 F-2 imblearn 安装界面图

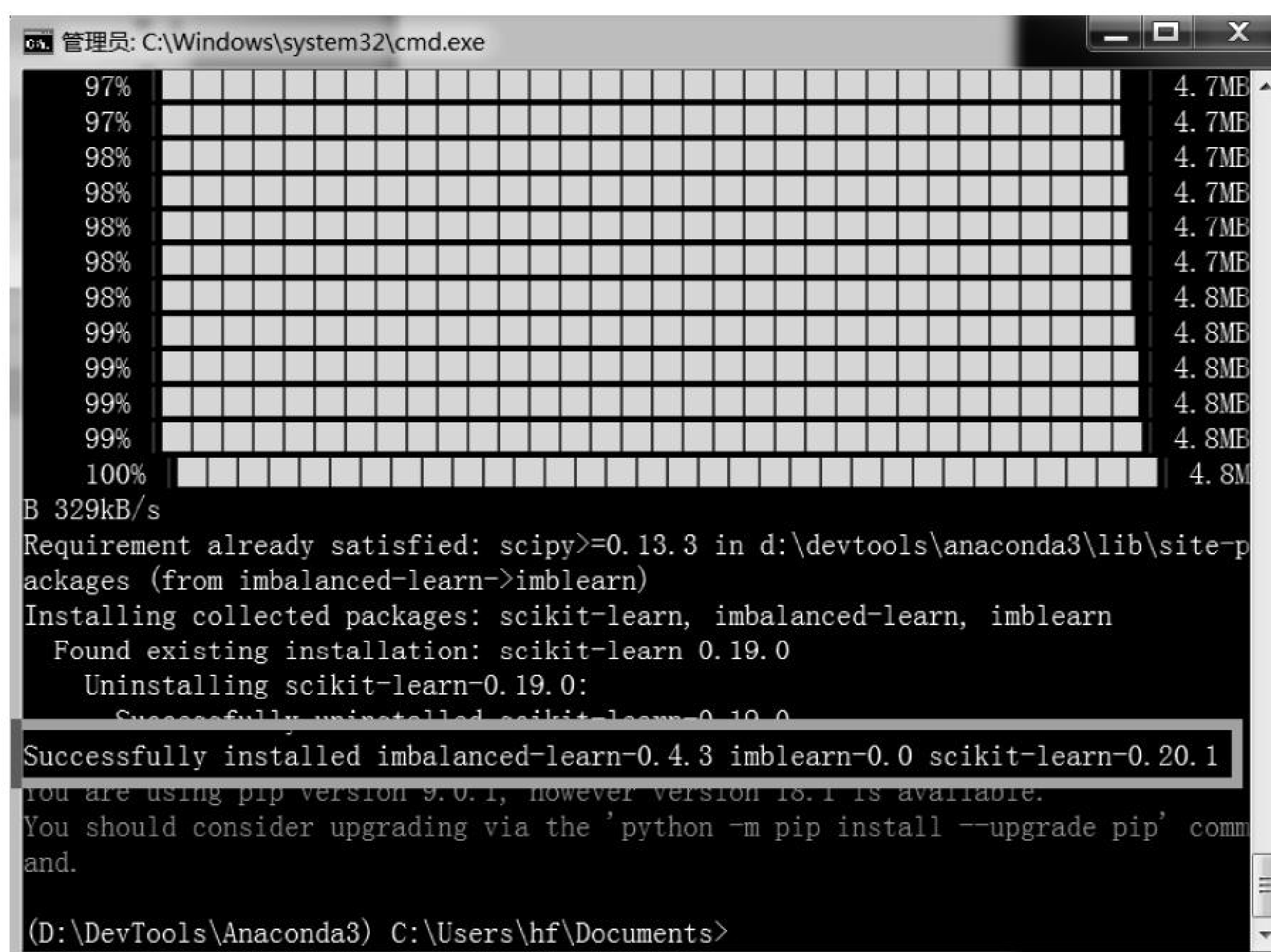


图 F-3 imblearn 安装成功界面图

图 书 资 源 支 持

感谢您一直以来对清华版图书的支持和爱护。为了配合本书的使用，本书提供配套的资源，有需求的读者请扫描下方的“书圈”微信公众号二维码，在图书专区下载，也可以拨打电话或发送电子邮件咨询。

如果您在使用本书的过程中遇到了什么问题，或者有相关图书出版计划，也请您发邮件告诉我们，以便我们更好地为您服务。

我们的联系方式：

地　　址：北京市海淀区双清路学研大厦 A 座 707

邮　　编：100084

电　　话：010－62770175－4520

资源下载：http://www.tup.com.cn

电子邮件：huangzh@tup.tsinghua.edu.cn

QQ：81283175（请写明您的单位和姓名）



书圈

用微信扫一扫右边的二维码，即可关注清华大学出版社公众号“书圈”。